TECNICAS EXACTAS Y HEURISTICAS EN EL DISEÑO AUTOMATICO DE REDES DE PLANIFICACION DE PROYECTOS

TESIS DOCTORAL **Rafael Martí Cunquero**

Departamento de Estadística e Investigación Operativa. Universitat de València. Abril 1993

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

→ MATEMATICAS

30 cms.

D. Vicente Valls Verdejo, profesor titular del Departamento de Estadítica e Investigación Operativa de la Universitat de València.

Certifica: Que la presente memoria "Técnicas exactas y heurísticas en el diseño automático de redes de planificación de proyectos" ha sido realizada bajo su dirección, en el Dpto de Estadística e Investigación Operativa por Rafael Martí Cunquero, y constituye su tesis para optar al grado de Doctor en Ciencias Matemáticas.

Y para que conste, en cumplimiento de la legislación vigente, presenta ante la Facultad de Matemáticas de las Universitat de València, a 29 de Abril de 1993.

El Director

Fdo: Vicente Valls Verdejo.

A Mila, Jose y Vicente,

por su dedicación al estudio.

Indice

Sección II: Grafos Acíclicos

Una de las necesidades básicas de los departamentos de planificación de empresas dedicados a proyectos de grandes dimensiones es la de contar con planos o redes capaces de representarlos con la mayor claridad posible, lo que se inscribe en el campo del dibujo de los grafos dirigidos acíclicos. El objetivo de esta memoria es investigar algunos aspectos esenciales de esta necesidad y ofrecer soluciones.

Para modelizar sistemas complejos se utilizan comunmente grafos dirigidos acíclicos, cuyos vértices representan los elementos de cada sistema y las aristas las relaciones entre sus elementos. Son una herramienta fundamental para el planteamiento y resolución de problemas de grandes dimensiones, como lo prueban las investigaciones realizadas en diversos campos: Proyectos PERT, Interpretación de modelos estructurales (I.S.M.) (Warfield [53]), Generación automática de diagramas lógicos (Smith y Linders [40]), Grafos de llamadas de subrutinas en programación de algoritmos, Diagramas de organización, Layout: extensión a grafos no acíclicos (Leung [32]), Diseño de sistemas de información (Tamassia [48]).

En todos los casos el personal responsable del desarrollo del sistema necesita a diario planos o dibujos que representen su estructura. La experiencia dice que un dibujo claro y comprensible del grafo constituye una ayuda de inestimable valor para lograrlo.

El diseño de estos planos es un problema muy difícil pues la posición de unos vértices condiciona la de otros, y lo mismo sucede con el trazado de las aristas. La cuestión se agrava más en la práctica porque los sistemas a modelizar contienen muchos elementos y, por lo tanto, los grafos resultantes son de gran tamaño.

Existe una dificultad añadida porque la mayoría de las veces el primer grafo dibujado no es el documento final, pues la modelización del sistema ha de pasar por la construcción de sucesivos planos que permitan interpretarlo mejor mediante diversas rectificaciones y diseños. Y todo ello, para que sea operativo, en el menor tiempo posible.

Por todas estas razones es extremadamente difícil poder diseñar y dibujar manualmente buenos planos de sistemas, lo que hace necesario construir un procedimiento automatizado para hacerlo a partir únicamente de sus elementos y de las relaciones que los definan.

Pese a que la memoria se ha realizado considerando el problema específico del diseño y dibujo de grafos PERT/CPM en la planificación de proyectos, las técnicas desarrolladas se pueden aplicar, en la mayoría de los casos, a cualquiera de los campos de investigación citados anteriormente.

El **objetivo de esta investigación**, en síntesis, es: **obtener de manera automática una representación gráfica, lo más clara y operativa posible, de una red de actividades definida mediante una lista de adyacencia.**

Aunque debemos admitir que el enunciado de dicho objetivo es impreciso a nivel teórico, pues debería definirse qué se entiende por una representación gráfica clara y operativa, a nivel práctico y de manera intuitiva resulta fácil determinar cuando un plano es "bueno" y cuando no lo es. Así, el siguiente ejemplo muestra como la claridad de la representación de un proyecto depende de la forma del dibujo. En la figura 1 es difícil extraer información sobre el proyecto, mientras que en la figura 2 su estructura se percibe a simple vista.

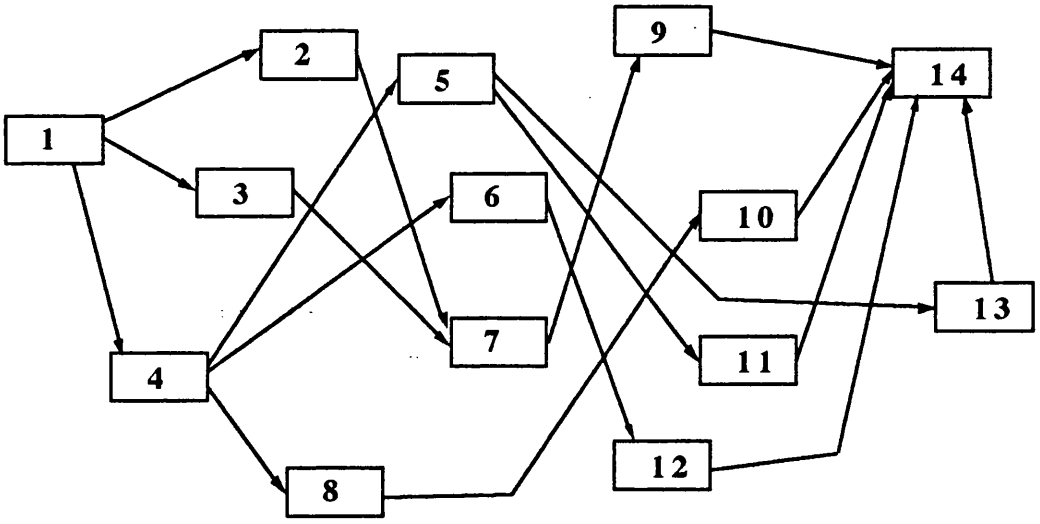


Figura 1

Pese a que las figuras 1 y 2 representan el mismo grafo de catorce vértices y dieciocho aristas, la segunda permite una interpretación más clara por la distribución más operativa de sus vértices y el trazado de las aristas.

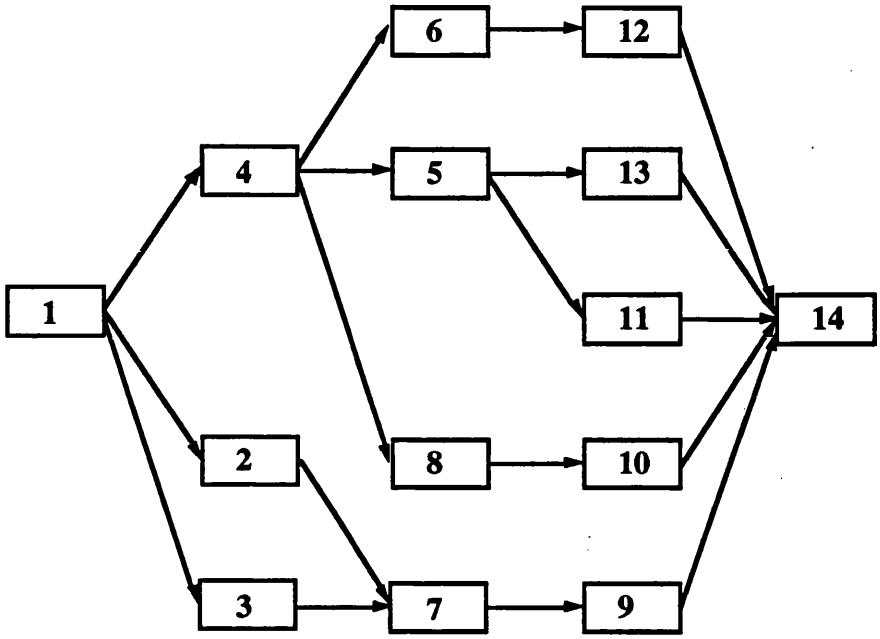


Figura 2

Un paso necesario para lograr el objetivo previsto es sistematizar el proceso de ubicación de los vértices y el trazado de las aristas, de manera que el resultado sea claro y legible.

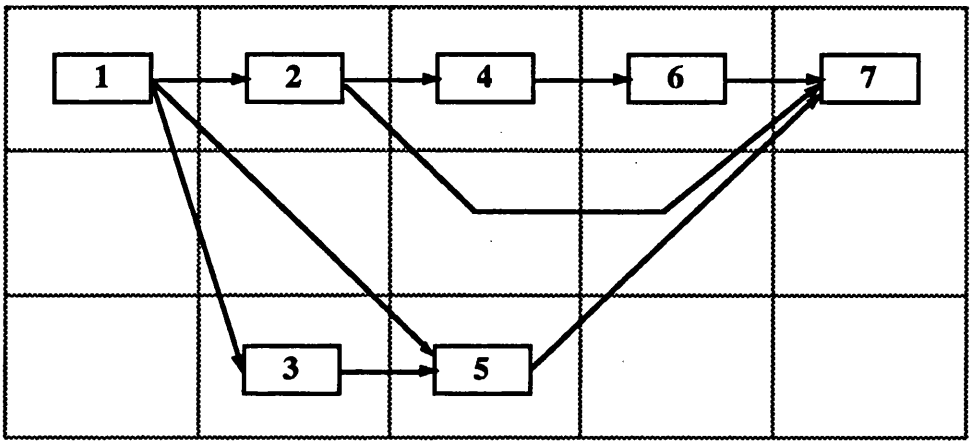
El problema abordado en esta memoria es de los que se denominan poco estructurados ("ill structured"), al no contar con un modelo claramente definido: función o funciones objetivo a optimizar, restricciones o acotaciones del problema,....Por ello, antes de resolverlo hemos de ir dándole una estructura. El primer paso, a tal efecto, es fijar unos criterios concretos que expresen claridad y operatividad, y permitan, al mismo tiempo, medir la bondad de las soluciones.

Para simplificar y discretizar el problema de la situación de los vértices hemos adoptado el criterio de ubicarlos en una cuadrícula o plantilla. Es el procedimiento que utilizan Warfield [53], Carpano [3] y otros autores.

En algunos de los trabajos publicados sobre representación de grafos, el trazado de las aristas juega un papel determinante al permitir cualquier tipo de recorrido: circular, lineal, poligonal,... . Este suele ser el caso de los algoritmos que construyen una representación sin cruces de un grafo planar, como por ejemplo el trabajo de Hopcroft y Tarjan [23].

La mayor parte de los autores, que consideran como objetivo principal la claridad y operatividad del grafo, asumen la restricción de que todas las aristas sean dibujadas mediante segmentos rectilíneos. Este es el caso de Sugiyama y otros [1] o Eades y Kelly [9]. Coincidimos con ellos en que el trazado de las aristas mediante cualquier tipo de línea puede proporcionar una solución con poco valor práctico u operativo.

De este modo, en adelante los vértices se situarán en las casillas de una cuadrícula y las aristas se dibujarán exclusivamente mediante segmentos rectilíneos, o con una poligonal de tres lados, tal y como muestra el siguiente ejemplo:



Tras analizar diferentes dibujos de grafos de proyectos hemos extraído los siguientes criterios que, a nuestro entender, proporcionan representaciones claras y operativas:

- Trazar las aristas de izquierda a derecha, nunca de derecha a izquierda. (Martí [34])
- Minimizar el número de cruces de aristas. (Eades y Kelly [9])
- Utilizar el mínimo número de columnas de la plantilla.
- Situar los vértices adyacentes en posiciones cercanas. (Sugiyama y otros[44])
- Reducir la suma de las longitudes de las aristas. (Eades y Wormald [13], Tagawa [45])
- Dibujar claramente las estructuras de ramificación y unión. (Sugiyama y otros [44])
- Evitar una excesiva densidad de información. (Carpano [3])

Los algoritmos desarrollados en esta memoria se han guiado por estos criterios, llegándose a un compromiso cuando han entrado en conflicto. Como todos los autores convienen en señalar que la minimización del número de cruces de las aristas es el más

importante de dichos criterios, se le ha dedicado una especial atención al principio de nuestro estudio.

En el **primer capítulo** se estudian los antecedentes del problema en la literatura científica y se comentan brevemente algunos conceptos previos de investigación operativa que serán utilizados a lo largo de la memoria.

El trabajo desarrollado se ha dividido en dos partes.

La **primera parte**: "Minimización del número de intersecciones de las aristas de un grafo acíclico", comprende los capítulos segundo y tercero. Dada la complejidad que implica este tema, comenzamos estudiando en la primera sección del **segundo capítulo** el caso particular de grafos bipartidos, que aún siendo más simple, es NP-completo (Eades [11]). Así, se presenta un algoritmo exacto basado en las técnicas de "Branch & Bound", que según nuestras referencias bibliográficas, es el primer algoritmo exacto diseñado para resolver el problema planteado.

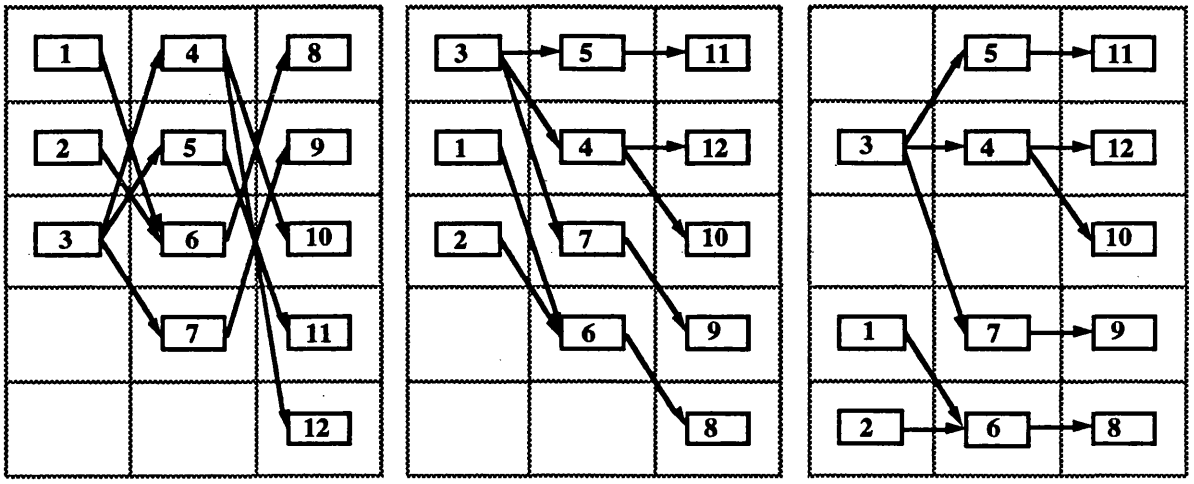
En la segunda sección del capítulo, se generaliza el algoritmo exacto a grafos acíclicos cualesquiera. Sin embargo, tal generalización incrementa en gran medida el número de soluciones posibles, haciendo impracticable resolver problemas reales, de gran tamaño, de manera exacta.

Para resolver el problema anterior y dar una buena solución inicial al algoritmo de ramificación y acotación presentado anteriormente, en el **capítulo tercero**, se desarrolla un algoritmo heurístico basado en las técnicas "Tabu Search" que minimiza el número de intersecciones de las aristas de grafos acíclicos cualesquiera, proporcionando buenas soluciones con bajos tiempos de computación; además permitirá incorporar fácilmente otros criterios de dibujo y será la base para desarrollar los algoritmos de representación de grafos de la segunda parte de la memoria.

Para medir la eficiencia del algoritmo heurístico se muestra una tabla de resultados, donde se compara con los más eficientes de la literatura científica y con el algoritmo exacto presentado en capítulos anteriores.

La **segunda parte**: "Diseño y representación de planos de proyectos", comprende los **capítulos** 4 y 5, dedicados al problema del dibujo de un grafo dirigido acíclico. En el **cuarto**, se estudian en primer lugar los criterios que le proporcionarán claridad y

operatividad. En él se presenta un algoritmo que obtiene una representación automática del grafo en tres fases, que refleja el siguiente ejemplo:



Fase 0: Jerarquización. Fase 1: Minimización de intersecciones. Fase 2: Distribución final.

En la fase 0 se estudia la asignación de los vértices del grafo a las columnas de la plantilla, transformando de este modo el grafo en una jerarquía. Para respetar el criterio de que las aristas sean trazadas de izquierda a derecha, en dicha asignación se considera que situar un vértice en una columna implica también ubicar sus sucesores en columnas posteriores. Con esta restricción se plantea la asignación de modo que, el número de columnas de la plantilla necesarias para ubicar todos los vértices del grafo sea mínimo.

Al realizar la asignación existen algunos vértices que únicamente pueden ser situados en una columna y otros que lo pueden ser en varias. Considerando esta holgura, realizamos la asignación de modo que la suma de las longitudes de las aristas sea mínima[1], lo que se resuelve de manera óptima mediante una formulación cuyo problema dual es de flujo de coste mínimo.

Una vez situados todos los vértices en columnas, para simplificar el trazado de las aristas, se sustituyen aquellas que unen vértices situados en columnas no consecutivas, por cadenas de vértices ficticios y aristas, de manera que todas las resultantes unan vértices de columnas consecutivas. Al aplicar el algoritmo de minimización de intersecciones de aristas visto en el capítulo cuarto al grafo obtenido, se reordenan los vértices. Al sustituir en la solución del algoritmo las cadenas de vértices por las aristas

[1] Se considera que la longitud de una arista es la diferencia entre número de columna de su vértice final y el inicial.

originales pueden aparecer trazados muy irregulares, por ello, en la fase 1 se modifica el algoritmo heurístico del capítulo cuarto introduciendo dos restricciones: 1) todos los vértices ficticios que sustituyen a la misma arista original han de estar en la misma fila de la plantilla, y 2) dicha fila deberá ser cercana a las ocupadas por los vértices extremos originales de la arista. Tales restricciones introducen bastante complejidad en el conjunto de movimientos que se consideran en cada paso del algoritmo "Tabu".

En la fase 2 se reubican los vértices de cada columna sin alterar las ordenaciones obtenidas en la anterior, de modo que se dibujen claramente las estructuras de ramificación y unión y se reduzca la suma total de las longitudes de las aristas[2].

Los criterios utilizados hasta el momento tienen un ámbito de aplicación más amplio que el de planos de proyectos. Sin embargo, los gestores de estos últimos tienen preferencias específicas dictadas por una utilidad práctica:

- Situar una secuencia de actividades "importantes" en el centro del dibujo de manera que sugiera una "escala temporal" del desarrollo del proyecto.
- Situar un grupo de actividades con características comunes en una misma región del plano.
- Permitir expandir un vértice (o colapsar un grupo de vértices) en el proceso de refinamiento y diseño de un proyecto, sin alterar gravemente la apariencia del plano.

En el **capítulo quinto** se desarrolla una metodología que, al trabajar con los conceptos de eje y componentes, permite incorporar las preferencias específicas mencionadas en el dibujo automatizado de planos de proyectos.

En el apartado de **conclusiones** se muestran los resultados obtenidos en esta memoria así como las futuras lineas de investigación.

[2]En este caso la longitud de una arista es la euclídea y no la simplificación considerada en la fase 0.

CAPITULO I

Preliminares

1. NOMENCLATURA Y TERMINOLOGIA.

En este apartado se describen las definiciones y conceptos básicos de la *Teoría de Grafos*. Si se desea profundizar en estos conceptos se pueden consultar Bondy y Murty [2] y Christofides [5].

A continuación, comentaremos brevemente algunas de las definiciones básicas que se utilizan en esta memoria.

Un *Grafo* G es una terna ordenada $(V(G),A(G),\Psi_G)$, donde V(G) es un conjunto, no vacío de *vértices*, A(G) es un conjunto de *arcos* y Ψ_G es una *función de incidencia* que asocia a cada arco de G un par no ordenado de vértices de G. Si e es un arco y u y v son vértices tales que $\Psi_G(e) = \{u,v\}$, se dice que u y v son los extremos del arco e. Para simplificar la notación un grafo será denotado por G=(V,A).

Un *bucle* es una arista cuyos dos extremos son idénticos.

Un grafo es *simple* si no tiene bucles y no hay dos arcos que unan el mismo par de vértices. En lo sucesivo, cuando hablemos de grafos, nos estaremos refiriendo a grafos simples, si no se dice explícitamente lo contrario.

El *grado de un vértice* v es el número de arcos de G incidentes con ese vértice. Lo denotaremos por d(v).

Un *camino* en G es una sucesión finita no nula $P=v_0a_1v_1a_2 \ldots a_kv_k$, cuyos elementos son alternativamente vértices y arcos, de forma que, para $1 \leq i \leq k$, los extremos de a_i son v_{i-1} y v_i. Los vértices v_0 y v_k son, respectivamente el origen y final del camino. En un grafo simple, un camino está determinado por la secuencia de sus vértices. Por ello un camino en un grafo simple se denotará únicamente mediante la secuencia de sus vértices.

Un *ciclo* es un camino en el que coinciden los vértices inicial y final.

Un *Grafo dirigido* G es una terna ordenada $(V(G),E(G),\Psi_G)$, donde V(G) es un conjunto, no vacío de *vértices*, E(G) es un conjunto de *aristas* y Ψ_G es una *función de incidencia* que asocia a cada arista de G un par ordenado de vértices de G. Si e es una

arista y u y v son vértices tales que Ψ_G (e) = {u,v}, se dice que u y v son, respectivamente el origen y el final de la arista e.

Un *camino dirigido,* en un grafo dirigido G, es una sucesión finita no nula $W=v_0a_1v_1a_2 \ldots a_kv_k$, cuyos elementos son alternativamente vértices y aristas, de forma que, para $1 \leq i \leq k$, la arista a_i tiene su origen en v_{i-1} y su final en v_i.

Un *circuito* es un camino dirigido en el que coinciden los vértices inicial y final.

Un *camino Hamiltoniano* en un grafo dirigido G es un camino dirigido que contiene todos los vértices de G.

Un grafo es *bipartido* cuando su conjunto de vértices puede ser particionado en dos subconjuntos X e Y de manera que, toda arista tiene un extremo en X y el otro en Y.

En todo el trabajo posterior usaremos grafos dirigidos. Así pues, cuando hablemos de grafos, estaremos refiriéndonos a grafos dirigidos, a menos que se especifique lo contrario.

Los conceptos y algoritmos fundamentales de la teoría de *Flujos en Redes* utilizados en esta memoria pueden encontrarse en Kennington y Helgason [26] y Bazaraa y Jarvis [1]. Asimismo, en lo referente a *Secuenciación de Proyectos* pueden consultarse las siguientes referencias: Moder, Phillips y Davis [36] y Wiest y Levy [57].

1.1 TECNICAS DE OPTIMIZACION "TABU SEARCH".

Algunos de los algoritmos utilizados se acogen a la técnica denominada Simple Tabu Thresholding, la cual es una versión probabilista de las técnicas heurísticas Tabu Search (Glover [19] y [20]), probadas en gran variedad de problemas clásicos de optimización obteniendo buenos resultados. Estas técnicas se basan en la definición de un conjunto de movimientos que unidos a unas listas de candidatos permiten pasar de una solución a otra del problema guiados por unos evaluadores y unas restricciones que regulan la optimización e impiden el ciclado en dichos movimientos.

Las técnicas Tabu Search se basan en realizar movimientos sobre el conjunto de soluciones del problema de optimización; con lo cual, toda solución tiene asociados una serie de movimientos, cada uno de los cuales proporcionará una nueva solución al ser aplicado sobre la actual.

El proceso de optimización se divide en dos fases que son repetidas alternativamente hasta alcanzar una solución satisfactoria o hasta que el número de iteraciones alcance un valor establecido; éstas son **Intensificación** y **Diversificación**: En la primera se optimiza en una determinada región del conjunto de soluciones hasta encontrar el óptimo local, en la segunda se abandona éste y se dirige la búsqueda a otra zona del conjunto de soluciones. Este método de búsqueda permite contemplar soluciones que no cumplen todas las restricciones del problema.

Dada una solución x se consideran los conjuntos siguientes:

Move_set(x) = Conjunto de movimientos asociados a la solución x.
Tabu(x) = Conjunto de movimientos de Move_set(x) que están prohibidos y que se denominan **movimientos tabu.**
Aspire(x) = Conjunto de movimientos suficientemente atractivos como para ser considerados pese a ser Tabu.

Aplicando las **estrategias de listas de candidatos** al conjunto Move_set(x) mediante un evaluador que mida la "bondad" de los movimientos, se escoge el que transformará la solución actual. En la elección no se consideran aquellos que son Tabu, salvo si cumplen el **criterio de aspiración** (pertenecen a Aspire(x)).

Relacionados con las fases de búsqueda se encuentran los modelos de memoria que especifican condiciones para guiar dicha búsqueda. El modelo "Short Term Memory" la guía en la fase de Intensificación, y el modelo "Long Term Memory" hace lo propio a largo plazo en la fase de Diversificación. Ambos han de ser establecidos en cada problema concreto.

Los objetivos de los controles Tabu son el impedir el ciclado de la búsqueda e introducir mayor flexibilidad y "vigor" en la exploración.

Existe una versión sencilla de implementar dentro de estas técnicas generales de optimización conocida como Simple Tabu Thresholding (Glover [21]) en donde se introducen criterios probabilísticos en la elección de los movimientos sin referencia explícita a ninguna forma de memoria y donde se introducen listas de candidatos con el objeto de disminuir el esfuerzo computacional que supondría evaluar todos los movimientos asociados a una solución.

Las fases de Intensificación y Diversificación reciben aquí los nombres de **Improving** y **Mixed** respectivamente. El conjunto M es un conjunto de movimientos que particiona al conjunto Move_set(x). Así pues, tendremos que:

$$\text{Move\_set}(x) = \bigcup_{i \in M} \text{Move\_set}(i,x)$$

Veamos la descripción algorítmica de las dos fases citadas:

IMPROVING PHASE

Explorar el conjunto M según la estrategia Block-Random Order Scan:

Considerar M como una lista circular ordenada. Dividirla en bloques. Tomar en cada iteración un bloque examinando sus elementos de modo aleatorio. Una vez seleccionado un elemento de M, aplicar las estrategias de listas de candidatos a los movimientos asociados a dicho elemento.

Sólo se acepta un movimiento si mejora el valor de la función objetivo.

Terminar con un óptimo local.

MIXED PHASE

Tomar t ∈ [l,u] aleatoriamente donde l y u son dos cotas predeterminadas.

Explorar M con la estrategia Full-Random Order Scan:

Tomar en cada iteración un elemento de M de modo aleatorio. Una vez seleccionado, aplicar las estrategias de listas de candidatos a los movimientos asociados a dicho elemento, aceptando automáticamente el movimiento candidato.

Hacer t iteraciones o bien parar cuando un cierto criterio de aspiración sea satisfecho.

Al terminar esta segunda fase, se vuelve a la fase improving.

La división del conjunto de movimientos en bloques y la elección pseudoaleatoria de los elementos de dichos subconjuntos, emula el funcionamiento de una lista Tabu impidiendo el ciclado. Así mismo, a la hora de decidir la longitud de los intervalos es interesante escoger un valor que no divida al cardinal de M, puesto que de esta manera, al ser la lista circular los intervalos varían en cada exploración sucesiva de M.

El algoritmo termina tras un número prefijado de iteraciones de ambas fases o cuando se cumple algún criterio global.

2.- ANTECEDENTES AL PROBLEMA

La primera referencia que hemos encontrado en el contexto de dibujo y representación de grafos es el trabajo de Wagner [52] en 1936. Dicho autor prueba que todo grafo planar se puede representar en el plano de forma que sus aristas sean lineas rectas y no se intersecten.

El mismo resultado fue probado posteriormente, y de manera independiente, por Fary [14] en 1948 y Stein [41] en 1951.

En 1960, W.T. Tutte [50] define "representación convexa" de un grafo G como un dibujo de lineas rectas sobre el plano, de manera que, todas las fronteras de las caras son polígonos convexos.

Una representación convexa de un grafo divide el plano en un número finito de regiones, cada una de las cuales, es el interior o exterior de un polígono, estableciendose así las condiciones necesarias y suficientes para la existencia de representaciones convexas de un grafo simple G.

En 1963, Tutte [51] introduce la definición de grafo 3-conexo. Estudia bajo que supuestos un grafo 3-conexo es planar y, se discuten condiciones para la existencia de subgrafos de Kuratowski. Se muestra cómo obtener una representación convexa de un grafo 3-conexo sin subgrafos de Kuratowski resolviendo un conjunto de ecuaciones lineales. Por último, se estudia la representación en el plano del grafo dual.

En 1969, Harary [22] proporciona dos cotas superiores al número de intersecciones de las aristas de un grafo. Una para el caso de un grafo completo y otra para un grafo bipartido completo.

Sea K_p el grafo completo sobre p vértices y $K_{m,n}$ el grafo bipartido completo. Sea v(K) el número de intersecciones de las aristas de una representación del grafo K. Entonces:

$$v(K_p) \leq \frac{1}{4}\left[\frac{p}{2}\right]\left[\frac{p-1}{2}\right]\left[\frac{p-2}{2}\right]\left[\frac{p-3}{2}\right] \qquad v(K_{m,n}) \leq \left[\frac{m}{2}\right]\left[\frac{m-1}{2}\right]\left[\frac{n}{2}\right]\left[\frac{n-1}{2}\right]$$

En 1969, Klovstad [27] introduce la idea de que al trabajar con teoría de redes para modelizar relaciones/condiciones tecnológicas, la red tecnológica ha de construirse con un ordenador. El input para que el ordenador dibuje la red ha de ser las relaciones tecnológicas entre las actividades.

En dicho artículo, Klovstad muestra algunos aspectos que habrán de considerarse al construir un procedimiento de dibujo. Señala que las actividades pueden agruparse en "rangos", situando en el rango i aquellas que se pueden realizar una vez finalizadas las del rango i-1. En este sentido indica que es factible obtener una red tecnológica situando las actividades lo antes posible y otra, situándolas lo más tarde posible. De este modo introduce los conceptos de holgura tecnológica y camino crítico tecnológico, con los cuales muestra la conveniencia de construir redes en las que las duraciones de las tareas que representan los vértices sean más o menos iguales para poder equiparar los conceptos mencionados con los de holgura y criticidad en tiempo real.

En 1971, Hope [24] presenta un programa para dibujar grafos planares. El programa, escrito en Fortran, tiene como input el orden de los nodos y aristas del grafo y, las asignaciones de estos a unas determinadas regiones. En el artículo se detallan los aspectos computacionales, tales como: estructura de datos, hardware utilizado, paquete gráfico empleado,... .

En 1974, Hopcroft y Tarjan [23] estudian el problema de la planaridad de un grafo, presentando un algoritmo para determinar si un grafo es o no planar; y en caso afirmativo, construir la representación sin cruces de aristas. Sin embargo, dicho algoritmo no considera ningún tipo de estructura al representar el grafo y no dibuja necesariamente las aristas mediante lineas rectas.

En 1976, Smith y Linders [40] analizan la minimización del número de cruces en el contexto de la generación automática de diagramas lógicos. No obstante, al igual que en el trabajo anterior, las aristas no son dibujadas necesariamente por lineas rectas. Además, la reducción del número de intersecciones está basada parcialmente en el trazado de las aristas.

El primer antecedente encontrado, que trata específicamente el problema que nos ocupa, es el artículo de Warfield [53] en 1977. Warfield se centra en el ámbito denominado Interpretive Structural Modeling (ISM) y en este artículo, señala la importancia de un buen plano o dibujo ("map") que represente al modelo. Al tratar con

modelos de gran tamaño muestra la necesidad de disponer de una teoría que, implementada en una máquina, produzca buenos planos de manera automática.

En primer lugar introduce el concepto de jerarquía y el de jerarquía propia que a continuación detallamos:

Definición:

Una jerarquía de n niveles o capas (n-level hierarchy) con n>1 se define como un grafo dirigido (V,E) donde V es el conjunto de vértices y E el de aristas, cumpliendo las siguientes condiciones:

$$V = V_1 \cup V_2 \cup V_3 \cup ... \cup V_n \quad / \quad V_i \cap V_j = \emptyset.$$

$$\forall (v_i,v_j) \in E \,/\, v_i \in V_m \,,\, v_j \in V_n \text{ se tiene que } m < n.$$

Llamaremos a V_i *nivel* o *capa* i (layer) y a n la longitud de la jerarquía. En jerarquías de sólo dos capas, éstas reciben también el nombre de *lados*. Denotaremos a la jerarquía por G = (V,E,n).

Definición:

Dada una jerarquía se define la extensión o longitud de una arista como la diferencia entre el número del nivel del vértice final de la arista menos el número del nivel del vértice inicial de la misma.

Definición:

Una jerarquía G= (V,E,n) es propia (proper hierarchy) cuando cumple:

$$E = E_1 \cup E_2 \cup ... \cup E_{n-1} \quad \text{donde } E_i \cap E_j = \emptyset \;\; / \;\; E_j \subseteq V_j \times V_{j+1}$$

Dicho de otro modo: "Una jerarquía es propia cuando la extensión de todas sus aristas es uno".

Tras establecer estos conceptos, Warfield señala que si una jerarquía es impropia, basta con introducir vértices adicionales reemplazando a las aristas de longitud mayor que uno, para hacerla propia.

A continuación, Warfield propone un procedimiento matricial para resolver el problema de minimizar el número de cruces entre aristas de una jerarquía propia. Dicho procedimiento, tiene un interés únicamente teórico, ya que por construcción, su aplicabilidad está limitada a redes de tamaños pequeños. A tal efecto, define la matriz de

interconexión para cada par de niveles consecutivos de una jerarquía propia e introduce unas fórmulas para calcular el número de intersecciones de las aristas de la jerarquía propia a partir de la matriz que la representa.

En 1978, Otten y Wijk [37] consideran una representación de grafos denominada "horvert". En dicha representación, los vértices del grafo son segmentos horizontales y las aristas segmentos verticales. Se muestra un procedimiento para construir una representación de este tipo y se prueba que un grafo es planar, si y sólo si, admite una representación horvert.

En 1980, Carpano [3] estudia los grafos jerarquizados, contrastando las definiciones de jerarquía dada por Warfield y las de otros autores (representación condensada, centrífuga, ...). Asimismo, extiende el concepto de jerarquía a grafos con ciclos y propone representaciones en tres dimensiones. Señala que en cualquier tipo de representación, un buen dibujo ha de cumplir dos condiciones:

- Número reducido de intersecciones de aristas.
- No debe haber demasiada densidad de información.

El autor propone un algoritmo para minimizar el número de intersecciones de las aristas en una jerarquía de n niveles (según la definición de Warfield [53]) para grafos acíclicos. El algoritmo ordena los vértices de un nivel, considerando fija la ordenación del nivel anterior. La ordenación se realiza según el principio: cuanto más horizontales sean las aristas menos cruces habrá. Así, obtiene la ordenada de un vértice como la media aritmética de sus vértices adyacentes. Sin embargo, no se muestran resultados computacionales

En 1981, Sugiyama y otros [44] tratan el problema de dibujar el plano de una jerarquía propia, para lo cual en primer lugar proponen los siguientes criterios para identificar una buena representación de la jerarquía (a readable map):

- Pocos cruces de aristas.
- Dibujar las aristas mediante líneas rectas.
- Situar los vértices adyacentes en posiciones cercanas.
- Dibujar claramente las estructuras de ramificación y unión.

Proporcionan unas reglas para la ubicación de los vértices y el trazado de las aristas en el dibujo del grafo, y proponen un método para reducir un grafo con ciclos a uno acíclico. Los autores formulan el dibujo del grafo como un problema multiobjetivo en varias etapas. La problemática de cada una de ellas es abordada en diversas aproximaciones teóricas, y su resolución, viene determinada por una serie de algoritmos heurísticos.

Los autores proponen dos algoritmos para minimizar el número de intersecciones de las aristas. El primero es una extensión del formulado por Warfield en [53] y, al igual que éste, su interés es sólo teórico. El segundo, está basado en los baricentros de las filas de la matriz de interconexión. Para medir la eficacia del algoritmo, se proporcionan unos parámetros sobre una colección de ejemplos aleatoriamente generados. El tamaño máximo de los ejemplos mostrados es de dieciséis vértices.

Los autores señalan la dificultad existente al sustituir una arista de longitud mayor que uno por vértices ficticios, ya que, al reordenar dichos vértices y sustituirlos por las aristas originales pueden aparecer trazados muy irregulares. Para resolver este inconveniente, introducen en el modelo la restricción de que todos los vértices ficticios que provienen de una arista original, han de estar alineados en la solución final. Sin embargo, hemos comprobado que, los algoritmos presentados violan, en algunos casos, dicha restricción.

En 1982, Johnson [25] prueba que, la minimización del número de intersecciones de las aristas en una jerarquía propia de dos niveles es un problema NP-completo.

Dentro del contexto de representación de grafos existe un problema frecuentemente estudiado conocido como "embedding a graph in a planar grid". Una de sus versiones es, dado un grafo planar G, situar cada uno de sus vértices en las intersecciones de una cuadrícula o reja plana y, dibujar sus aristas sobre las líneas horizontales y verticales de la cuadrícula.

Otros trabajos destacables son los de Rosemberg [39] en 1979, Fischer y Paterson [15] en 1980 o Floyd y Ullman [16] en 1980. En ellos se presentan algoritmos para resolver la problemática citada. Para medir la bondad de los algoritmos se utilizan dos criterios. El primero es el área que ocupa el menor rectángulo de la cuadrícula que contiene al grafo. El segundo es la suma total de las longitudes de las aristas.

En 1984, Storer [43] establece un tercer criterio para medir la bondad de los algoritmos. Al dibujar las aristas del grafo sobre la cuadrícula, en ocasiones hay que "torcerlas". Esto se puede modelizar situando un vértice cada vez que la arista cambia de dirección. Poniendo de manifiesto que el objetivo es la minimización del número de vértices necesarios para representar el grafo en la cuadrícula (node cost measure). Tras justificar la aplicación de este criterio, Storer prueba que encontrar una representación que minimice este tercer criterio es un problema NP-completo y propone tres aproximaciones polinómicas.

En 1987, Tamassia [49] presenta un algoritmo que, utilizando técnicas de flujos en redes, proporciona la solución óptima del problema planteado por Storer [43] con complejidad $O(n^2 \log n)$.

En 1985, Tamassia y Tollis [46] y [47], consideran lo tratado por Otten y Wijk acerca de la representación horvert. Estos primeros, trabajan sobre el concepto de visibilidad. Así, dos segmentos paralelos de un conjunto son visibles cuando pueden ser unidos por un segmento ortogonal a ellos tal que no intersecte a ningún otro. Definen la representación visible de un grafo (visibility representation) coincidiendo con la definición de representación horvert dada por Otten y Wijk. En ambos trabajos se consideran tres tipos diferentes de representaciones visibles y se caracterizan las clases de grafos que admiten ser representados de tales maneras. Además se presentan algoritmos que, en tiempos lineales, determinan la existencia o no de una representación visible de un grafo planar dado.

En 1988, Di Battista y Tamassia [7] estudian la representación de grafos acíclicos de manera que el flujo de todas las aristas circule en la misma dirección (monotonic drawing). Consideran tres posibles representaciones:

1 - Visibility Representation: Comentada anteriormente.
2 - Grid Drawing: Considerada por Storer [43] y otros.
3 - Straight (upward) Drawing: En el que todas las aristas están orientadas de abajo hacia arriba.

Los autores prueban que un grafo admite cualquiera de las tres representaciones si, y sólo si, es un subgrafo de un s-t grafo planar[3]. Asimismo presentan algoritmos para construir dichas representaciones.

En el mismo año, Tamassia y otros [48] hacen una revisión de los resultados publicados sobre el dibujo automático de grafos ("The state of the art"). Los autores categorizan los algoritmos estudiados según los siguientes parámetros: Clases de grafos (arboles, planares, dirigidos y no dirigidos), Tipo de representación [7], Criterios estéticos (Nº de cruces, longitud de las aristas,...), Restricciones de dibujo y Complejidad computacional. Por último, presentan un algoritmo para dibujar grafos mixtos.

El trabajo más importante, a nuestro entender, respecto al problema específico de minimizar el número de intersecciones de las aristas de un grafo, es el artículo publicado en 1986 por Eades y Kelly [9], en el cual proponen cuatro algoritmos heurísticos que reordenando los vértices, minimizan el número de cruces de aristas en jerarquías de dos niveles.

Los algoritmos son: Greedy Switching, Splitting, Averaging y Greedy Insertion. Como comentan los autores, el algoritmo Greedy Insertion es el que peor se comporta. En los grafos de baja densidad el que mejor funciona es el Splitting, y en los de densidad media-alta tanto el Splitting como el Averaging o Greedy Switching, producen buenos resultados, pudiendo ser el mejor este último. Veamos la terminología descrita por estos autores así como los algoritmos Greedy Switching y Splitting .

Notaremos a una red-jerarquía con dos niveles como G=(L,R,E), donde L es el conjunto de vértices del nivel izquierdo, R los del nivel derecho y E es el conjunto de aristas (está contenido en L x R.).

La minimización del número de intersecciones de las aristas de E equivale a encontrar una ordenación de los vértices de L y R, puesto que al ser todas las aristas líneas rectas las intersecciones de éstos únicamente dependen de la posición del vértice inicial y final.

---

[3] Un s-t grafo es un grafo acíclico con exactamente un vértice fuente s y un sumidero t y contiene el arco (s,t).

Una **ordenación** l del conjunto L es una función uno a uno de manera que a cada elemento de L le asigna un entero entre 1 y cardinal de L:

Un **dibujo** (l,r) de una red con dos niveles es una ordenación l del conjunto L de vértices y una ordenación r del conjunto R de vértices.

Un **cruce** es un conjunto $\{ (u,v) , (w,x) \} \subseteq E$ de dos aristas donde:

1. $u,w \in L$ y $v,x \in R$
2. $l(u) < l(w)$ y $r(v) > r(x)$ o, $l(u) > l(w)$ y $r(v) < r(x)$:

Un dibujo es **óptimo** si ningún otro tiene menos cruces.

Un dibujo (l,r) es **óptimo izquierdo** si no existe otra ordenación l' de L, de manera que (l',r) tenga menos cruces. Un dibujo (l,r) es **óptimo derecho** si no existe otra ordenación r' de R, de manera que (r',l) tenga menos cruces.

A continuación, pasamos a describir el evaluador que utilizan los autores citados para medir el número de intersecciones generadas por dos vértices en la posición en la que se encuentran.

Sea G=(L,R,E) una red con dos niveles, sea r una ordenación de R y l una ordenación de L. Dados dos vértices u,v en L se define c(u,v) como el número de cruces de aristas incidentes con u con aristas incidentes con v considerando que u precede a v en la ordenación l y que la ordenación r es fija.

$$c(u,v) = \left| \{ ((u,w) , (v,x)) \subseteq E \ / \ r(w)>r(x) \} \right| \quad \text{siendo } l(u) < l(v).$$

Algoritmo GREEDY SWITCHING

El algoritmo considera los conjuntos L y R con un orden inicial cualquiera. Toma el conjunto L con el orden l y, considerando el conjunto R con el orden r fijo explora L de arriba a abajo haciendo una permutación entre dos vértices consecutivos cuando esto disminuye el número de intersecciones:

Si u,v son dos vértices consecutivos de L ($l(u)=l(v)+1$) y $c(u,v)<c(v,u)$ entonces se intercambian las posiciones de u y v, con lo que el número de intersecciones del grafo disminuye en $c(v,u)-c(u,v)$.

Una vez explorado el conjunto L y realizadas las permutaciones, sea l el orden final de los vértices.A continuación se explora R considerando L con el orden l fijo.

El algoritmo itera sucesivamente sobre L y R y se detiene cuando ya no puede disminuir más el número de intersecciones.

Algoritmo SPLITTING

El algoritmo considera los conjuntos L y R con un orden inicial cualquiera. Toma el conjunto L y considerando el conjunto R con el orden r fijo se elige un vértice u de L como pivote. El resto de vértices de L se sitúan arriba o abajo de u según donde produzcan menos intersecciones. Este procedimiento se aplica recursivamente para ordenar los vértices situados arriba de u y los situados abajo.

Una vez ordenado el conjunto L sea l el orden final de los vértices, a continuación se ordena R considerando L con el orden l fijo.

El algoritmo itera sucesivamente sobre L y R y se detiene cuando ya no puede disminuir más el número de intersecciones.

Como se puede ver en el ejemplo dado en [9] la condición de óptimo izquierdo y de óptimo derecho no implica necesariamente la condición de óptimo. Sin embargo tanto los algoritmos presentados por Warfield [53], Sugiyama [44] como los de Eades y Kelly citados, buscan una solución que sea óptima izquierda y óptima derecha.

P. Eades y otros [11] prueban en 1986 que el problema de encontrar el óptimo derecho (análogamente izquierdo) es NP-completo.

En el mismo año, P. Eades y N.C. Wormald [13] dan a conocer un algoritmo denominado "Mediam heuristic" que reordena los vértices del nivel izquierdo de una jerarquía de dos niveles con dos objetivos:

- Reducir el número de intersecciones de las aristas.
- Reducir la suma de las longitudes de todas las aristas.

Los autores prueban que, el número de cruces de la solución proporcionada por el algoritmo, no excede de tres veces el número de cruces de la solución óptima izquierda.

En 1991, el mismo autor que el de este proyecto de tesis doctoral, en la memoria de licenciatura [34] trata el problema de representar de modo automático los vértices del grafo de un proyecto de actividades.

Puesto que el problema es de los llamados Poco Estructurados, el trabajo se divide en dos fases: En la primera se trabaja en establecer los criterios o propiedades que distinguen a un "buen dibujo", en el sentido operativo de la expresión; en la segunda se traducen estos criterios a algoritmos que, con una implementación eficiente, proporcionen la solución: distribución de los vértices sobre el plano.

En la **primera fase** se establecen como criterios generales o propiedades fundamentales la CLARIDAD y la CANTIDAD DE INFORMACION, entendiendo por claridad el que la red sea "fácilmente comprensible" y por cantidad de información el que queden dibujadas la mayor cantidad de aristas del grafo.

Concretando estas propiedades generales en criterios operativos, se considera en primer lugar y para simplificar el problema de la situación de los vértices sobre el papel el definir una plantilla o cuadrícula de trabajo.

En esta plantilla, se denota como *capa* a las columnas y *línea* a las filas, estableciendo de esta manera unos ejes cartesianos que permitirán referenciar la posición de los vértices sobre el papel. Así mismo, se fijan los siguientes criterios concretos:

- Situar las aristas preferentemente de izquierda a derecha.
- Prohibir las aristas de derecha a izquierda.
- Admitir, además de las primeras, sólo aristas de arriba a abajo y de abajo a arriba.

Por último, y puesto que existe una relación evidente entre la situación de los vértices y el trazado de las aristas, se enuncian las dos propiedades siguientes:

- Cuantas más aristas se representen horizontalmente, más claro será el dibujo.
- La red será más clara cuanto menor sea el número de intersecciones entre aristas.

En la **segunda fase** del trabajo se diseña una secuencia algorítmica guiada por los criterios de la primera fase que proporciona la distribución de los vértices de un grafo sobre la plantilla, de manera automática, a partir de una descripción del grafo mediante listas de sucesores y predecesores.

El algoritmo tiene las siguientes partes:

1.- Obtener el camino más largo que más desconecte el grafo.

Para comenzar a dibujar el plano se considera un eje que vertebre el dibujo: se toma el camino más largo puesto que cumple los criterios propuestos. Dado que pueden aparecer varios caminos más largos en un grafo, se toma, de entre todos ellos, el que más desconecte dicho grafo al eliminar tal camino, con el objetivo de hacer más flexible el algoritmo.

Para realizar este apartado construimos en primer lugar el que denominamos grafo de caminos más largos mediante un algoritmo que calcula , en un paso hacia adelante, las distancias respecto al número de aristas del primer vértice a todos los demás y, en un paso hacia atrás, recupera los caminos más largos. Sobre este grafo, y mediante un algoritmo de flujo de coste mínimo, se calcula el camino con mayor suma del grado de sus vértice; es decir, el camino tal que, al eliminarlo del grafo, más lo desconectará. Llamamos P a tal camino.

2.- Calcular las componentes conexas al eliminar P.

En el grafo resultante al eliminar P de la red original aparecen una serie de componentes conexas que consideraremos como unidades de dibujo.

Para su cálculo se utiliza un algoritmo recursivo en el grafo original con unas etiquetas para señalar los vértices de cada componente y diferenciar los vértices del camino P con los restantes.

3.- Estructurar cada componente conexa.

Aplicando a cada componente el mismo razonamiento que al grafo total, se descompone cada componente en sucesivos caminos más largos (eliminándolos de la componente y calculando el camino más largo del grafo que queda) denominando "rectas" a cada uno de estos caminos.

Se simplifican varias rectas en una en los casos en que sea posible. Para maximizar el número de simplificaciones se utiliza el teorema de Dilworth definiendo un problema de flujo máximo sobre una red auxiliar. A cada uno de los caminos resultantes los denominamos super-rectas.

Por último, se ordenan las super-rectas de cada componente bajo el criterio de minimizar las intersecciones entre aristas.

4.-Asignar los vértices.

El algoritmo finaliza con la asignación de las componentes conexas en torno a P. Para ello se formaliza la hoja de trabajo mediante una matriz en la que cada fila y columna representa a una linea y capa. La distribución final se obtiene mediante un algoritmo que asigna secuencialmente las componentes en torno a P según los criterios propuestos en la primera fase.

La eficiencia del algoritmo es medida contrastando los resultados obtenidos por éste, con las rutinas de dibujo de los programas comerciales de planificación de proyectos que se encuentran en el mercado actual: Power Project, Microsoft Project para windows, Harvard Project manager y super project expert, constatando la superioridad del algoritmo de dibujo presentado.

En 1992, Michalewizc [35] presenta dos algoritmos genéticos para el dibujo de un grafo dirigido. Considerando los dos criterios propuestos por Eades y Lin [10], los algoritmos reordenan los vértices del grafo sobre las casillas de una plantilla. No se muestran resultados computacionales, únicamente dos ejemplos concretos.

Una aplicación práctica del problema general del dibujo de grafos, es la conocida como Facility Layout, en donde se han de establecer las posiciones de las máquinas en un taller de producción flexible. Así, entre otros trabajos, podemos señalar el de Leung [40] en 1992, donde resuelve un primer paso de este problema: Determinar qué maquinas han de estar en posiciones adyacentes. Para ello desarrolla un algoritmo heurístico que determina el máximo subgrafo planar de un grafo dado.

PRIMERA PARTE

Minimización del Número de Intersecciones de las Aristas de un Grafo Acíclico.

CAPITULO II

Algoritmo de Ramificación y Acotación.

En este capítulo tratamos el problema de minimizar el número de intersecciones generadas por las aristas de un grafo acíclico. El estudio se divide en dos secciones; en la primera, se estudia el caso de grafos bipartidos (jerarquías propias de dos niveles), en la segunda se extiende el estudio a grafos acíclicos cualesquiera.

Junto con los estudiosos del dibujo de grafos, consideramos en la **primera sección** por razones de simplicidad, claridad y operatividad, que las aristas del **grafo bipartido** se representan por el segmento rectilíneo que une los dos vértices adyacentes. De este modo, las intersecciones dependen exclusivamente de la disposición (ordenación) de los vértices, por lo que este problema es de naturaleza combinatoria. Eades y otros [11] prueban que este problema es NP-completo.

En el primer apartado de esta sección, se formula el problema como la minimización de una expresión cuadrática sujeta a restricciones lineales con variables binarias. En el segundo se muestra un algoritmo exacto que explora el conjunto de soluciones posibles mediante las técnicas de ramificación y acotación de nodos (Branch & Bound).

En la **segunda sección** se generaliza a **grafos acíclicos** el algoritmo exacto mostrado en la primera. Sin embargo, el interés de la generalización es puramente teórico y no práctico, dado el límite del tamaño de los grafos bipartidos para los que el algoritmo encuentra el óptimo en un tiempo razonable.

En el primer apartado, se muestra como todo grafo acíclico se puede transformar en una jerarquía propia. En los siguientes, se generaliza el algoritmo de la primera sección a jerarquías propias de n niveles.

PRIMERA SECCION: GRAFOS BIPARTIDOS.

1 FORMULACION DEL PROBLEMA.

1.1 CALCULO DEL NUMERO DE INTERSECCIONES.

Como señalan Eades y Kelly [9], el problema consiste en encontrar las ordenaciones l y r de los vértices de L y R que minimicen el número de intersecciones de las aristas del grafo.

Definición

Dada la jerarquía G=(L,R,E) y dados dos vértices i,j $\in$ L, para una ordenación r de R se define K(i,j) como el número de intersecciones de aristas que parten del vértice i con aristas que parten del vértice j, cuando el vértice i precede al j en la ordenación de L.

Notar que dados i,j $\in$ L, el valor de K(i,j) no depende de la posición del resto de vértices de L y sí de la ordenación de los vértices de R.

De manera análoga se define el parámetro K(i,j), i,j vértices de R, para una ordenación l de L.

TEOREMA 1

Dados dos vértices i,j en una misma capa del grafo bipartido G=(L,R,E), K(i,j)+K(j,i) es constante para cualquier ordenación de la otra capa.

Prueba

Dados dos vértices i,j de L y una ordenación de los vértices de R, considerando los pares de aristas donde una es incidente con i y otra con j, se tienen la siguiente clasificación:

Clase 1.- Que las dos aristas formen un cruce, con lo cual al intercambiar las posiciones de i,j, ya no se intersectarán.

Clase 2.- Que dichas aristas tengan el mismo vértice final, con lo cual no se intersectan ni en la posición actual ni al permutar dichos vértices.

Clase 3.- Que dichas aristas no se intersecten y no tengan el mismo vértice final, con lo cual al permutar las posiciones de i,j se intersectarán.

K(i,j) proporciona el número de pares de aristas citados que pertenecen a la clase 1 si i precede a j. Análogamente, K(j,i) indica el número de dichos pares pertenecientes a la clase 1 si j precede a i, o lo que es lo mismo, el número de los que pertenecen a la clase 3 si i precede a j. Las aristas de la clase 2 no producen intersecciones.

En todas las ordenaciones de R, los pares de aristas de la clase 2 no varían, por lo que el número total de elementos de las clases 1 y 3 permanece constante. Así pues, para cualquier ordenación de R, K(i,j)+K(j,i) es un invariante. ■

Pasamos a construir el algoritmo que proporciona los valores de K(u,v) y K(v,u), a partir de la clasificación del teorema 1. Sean u y v vértices de L y consideremos fija la ordenación r de los vértices de R.

Algoritmo 1. Cálculo de K(i,j)

```
INICIALIZACION.
Sean los vértices u,v de L / l(u)<l(v).
K(u,v) = 0.
K(v,u) = 0.

WHILE( Existen sucesores de u sin explorar)
{
        Sea u´∈R sucesor de u sin explorar.
        WHILE( Existen sucesores de v sin explorar)
        {
                Sea v´∈R sucesor de v sin explorar.
                IF  r(u´) < r(v´)
                                K(v,u) = K(v,u) +1.
                ELSE IF r(u´) > r(v´)
                                K(u,v) = K(u,v) + 1.
                etiquetar v´como explorado.
        }
        Etiquetar u´como explorado.
        Eliminar las etiquetas de los sucesores de v.
}
```

Para calcular los valores de K(u,v) y K(v,u) para u y v en R, basta con sustituir en el algoritmo anterior la ordenación r por l, y los sucesores por predecesores.

Definición

Dada la jerarquía G=(L,R,E) con unas ordenaciones l,r de L y R se define K(G) como el número total de intersecciones de las aristas de G con dichas ordenaciones.

K(G) se puede obtener como suma de los parámetros K(i,j) con l(i)<l(j) para todos los vértices i,j de L (o análogamente como suma de todos los K(i,j) con r(i)<r(j) para todos los i,j de R):

$$K(G) = \sum_{l(i)<l(j)} K(i,j)$$

Veamos en el siguiente ejemplo el cálculo de K(i,j):

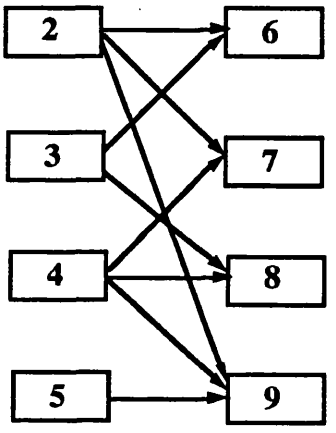


Figura 1

Tomemos, por ejemplo, los vértices 3 y 4 de L en donde l(3)=2<3=l(4), se tiene la siguiente clasificación de sus aristas incidentes según las clases introducidas anteriormente:

Clase 1.- {(3,8) (4,7)} luego K(3,4)=1

Clase 2.- {(3,8) (4,8)}

Clase 3.- {(3,6) (4,7)} {(3,6) (4,8)} {(3,6) (4,9)} {(3,8) (4,9)} luego K(4,3)=4

De este modo, considerando la ordenación mostrada en el ejemplo para el resto de parejas de vértices de L se obtiene:

K(G) = K(2,3)+K(2,4)+K(2,5)+K(3,4)+K(3,5)+K(4,5) = 3+2+0+1+0+0 = 6.

1.2 MATRIZ DE INTERCONEXION.

En [53] Warfield representa las ordenaciones de los vértices de una jerarquía de dos capas mediante una matriz que denomina matriz de interconexión. Basándose en esta matriz proporciona expresiones para evaluar el número de intersecciones de las aristas de la jerarquía y traduce el problema de reordenar los vértices de la jerarquía al de premultiplicar o postmultiplicar la matriz de interconexión por matrices de permutación.

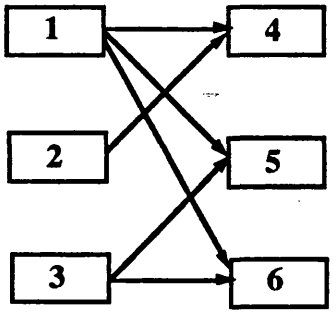
A partir de estas consideraciones se ha formulado el problema de minimizar las intersecciones de aristas mediante un problema de programación matemática con función objetivo cuadrática, con variables enteras con valores 0,1 y restricciones lineales.

Definición:

Dada una jerarquía de dos capas G=(L,R,E) y siendo l,r ordenaciones de los conjuntos L y R respectivamente, se define la matriz de interconexión N de dimensiones |L| * |R| asociada a las ordenaciones l y r como:

N(i,j) = 1 Si existe una arista del i-ésimo vértice de l al j-ésimo vértice de r.
N(i,j) = 0 En otro caso.

Ejemplo



$$N = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} 4 & 5 & 6 \end{array} \\ \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix} \end{array}$$

Figura 2

Sea la jerarquía G=(L,R,E) y la matriz de interconexión $N = (n_{ij})$ de dimensiones m*n de dicha jerarquía para unas ordenaciones l,r. Dados los vértices $i,j \in L / l(i)<l(j)$, el valor del parámetro K(i,j) se puede calcular de la siguiente manera:

$$K(i,j) = \sum_{k=1}^{n-1} \sum_{s=k+1}^{n} n_{l(j)k} * n_{l(i)s}$$

Como ilustra el grafo de la figura 2, el número de cruces de aristas que parten del vértice 1 con aristas que parten del vértice 2 es de: K(1,2)= 1*1 + 1*1 + 0*1 = 2.

A partir de esta expresión, para evaluar el número total de cruces del grafo basta con sumar para todos los posibles pares de vértices de L la expresión anterior:

$$K(G) = \sum_{l(i)<l(j)} K(i,j) = \sum_{l(i)<l(j)} \sum_{k=1}^{n-1} \sum_{s=k+1}^{n} n_{l(j)k} * n_{l(i)s} \qquad (1)$$

La expresión para evaluar el número de cruces es equivalente a sumar para cada 1 de la matriz el número de 1 que están en filas superiores y en columnas más a la derecha. Así, considerando el 1 en la posición (2,1) del ejemplo anterior se tiene que: el 1 en la posición (1,2) indica que las aristas (2,4) y (1,5) se intersectan y, el 1 en la posición (1,3) indica que las aristas (2,4) y (1,6) también se intersectan.

Sea G=(L,R,E) una jerarquía. Sean N_1 y N_2 las matrices de interconexión asociadas a las ordenaciones l,r y l',r' respectivamente. Puesto que N_2 se puede obtener de N_1 reordenando sus filas y columnas, existen matrices de permutación P y Q tales que $N_2 = P\ N_1\ Q$.

Ejemplo:

$$N_1 = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$

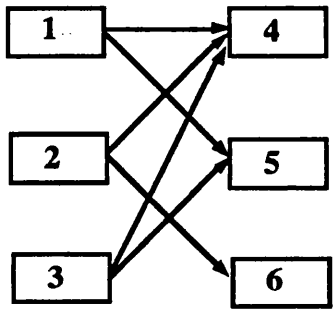


Figura 3

$$N_2 = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

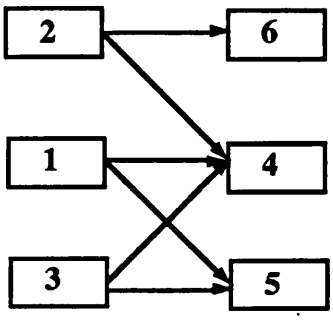


Figura 4

$$N_2 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} * N_1 * \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

El problema de minimizar el número de intersecciones en G, según la nomenclatura introducida, equivale a encontrar las matrices de permutación P y Q tales que la matriz PNQ minimice la expresión dada en (1), donde N es la matriz de interconexión inicial.

Así pues, se puede formular el problema de minimizar el número de intersecciones de un grafo bipartido representado por la matriz de interconexión N como:

$$\text{MINIMIZAR} \quad \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} \sum_{k=1}^{n-1} \sum_{s=k+1}^{n} p_j N q^k p_i N q^s$$

s.a.:

$$\sum_{i=1}^{m} p_{ij} = 1 \qquad j=1, \ldots ,m.$$

$$\sum_{j=1}^{m} p_{ij} = 1 \qquad i=1, \ldots ,m.$$

$$\sum_{i=1}^{n} q_{ij} = 1 \qquad j=1, \ldots ,n.$$

$$\sum_{j=1}^{n} q_{ij} = 1 \qquad i=1, \ldots ,n.$$

$$p_{ij} , q_{ij} = 0, 1 \qquad \forall i,j$$

donde $p_j = (p_{j1} , p_{j2} , \ldots , p_{jn}) \; \forall j$ y $q^k = (q_{1k} , q_{2k} , \ldots , q_{nk})^t \quad \forall k$

2 ALGORITMO DE RAMIFICACION Y ACOTACION.

El objetivo de este apartado es el de construir un algoritmo exacto basado en las técnicas de ramificación y acotación de nodos (Branch & Bound) para resolver el problema de minimizar el número de intersecciones de aristas formulado en el apartado anterior. Comenzaremos dando una descripción general del algoritmo en los subapartados: Arbol de soluciones, Cotas inferiores y Mejor solución, para después, en apartados posteriores, profundizar más en algunos aspectos e introducir criterios que proporcionen más eficiencia y rapidez al algoritmo.

2.1 ARBOL DE SOLUCIONES

Sea un grafo bipartido G=(L,R,E). Toda solución consta de dos ordenaciones (l,r): una ordenación l del nivel L y otra r del nivel R. El árbol de soluciones representa una manera de generar y particionar el conjunto de soluciones. Se construye así:

A partir del nodo inicial se ramifica en tantos nodos como ordenaciones posibles existen del nivel izquierdo: P_1, P_2, ..., $P_{n!}$ donde n es el número de vértices de L. P_i representa el conjunto de soluciones posibles en las que la ordenación del nivel izquierdo es l_i. Llamaremos a dichos nodos, "nodos del primer nivel de ramificación".

Cada nodo del primer nivel se ramifica en tantos nodos como vértices hay en el nivel derecho R. En cada uno de dichos nodos se fija el vértice de R que será el primero en la ordenación. Así pues, cada uno de dichos nodos representa el conjunto de soluciones posibles en las que la ordenación del nivel izquierdo es la fijada por el nodo padre y el primer vértice de la ordenación del nivel derecho es el fijado por el nodo. Llamaremos a dichos nodos: "nodos del segundo nivel de ramificación".

Cada nodo de un nivel de ramificación i con i>1 se ramifica en tantos nodos como vértices del nivel derecho queden por asignar en la ordenación r. En cada uno de los nodos hijos se fija el vértice que ocupa la posición i-1 en la ordenación del nivel derecho. De esta manera se va particionando el conjunto de soluciones posibles.

Notar que al asignar el vértice de R que será el penúltimo, queda fijada la posición del único restante, que necesariamente será el último. Por ello si en R hay n vértices, el árbol de soluciones tendrá n niveles.

El siguiente dibujo muestra una representación parcial del árbol de soluciones de una jerarquía de dos niveles con tres vértices en cada lado, L={1,2,3} y R={4,5,6}. En el árbol completo todos los nodos del primer nivel se ramifican en tres nodos y todos los del segundo nivel en dos nodos.

Ejemplo:

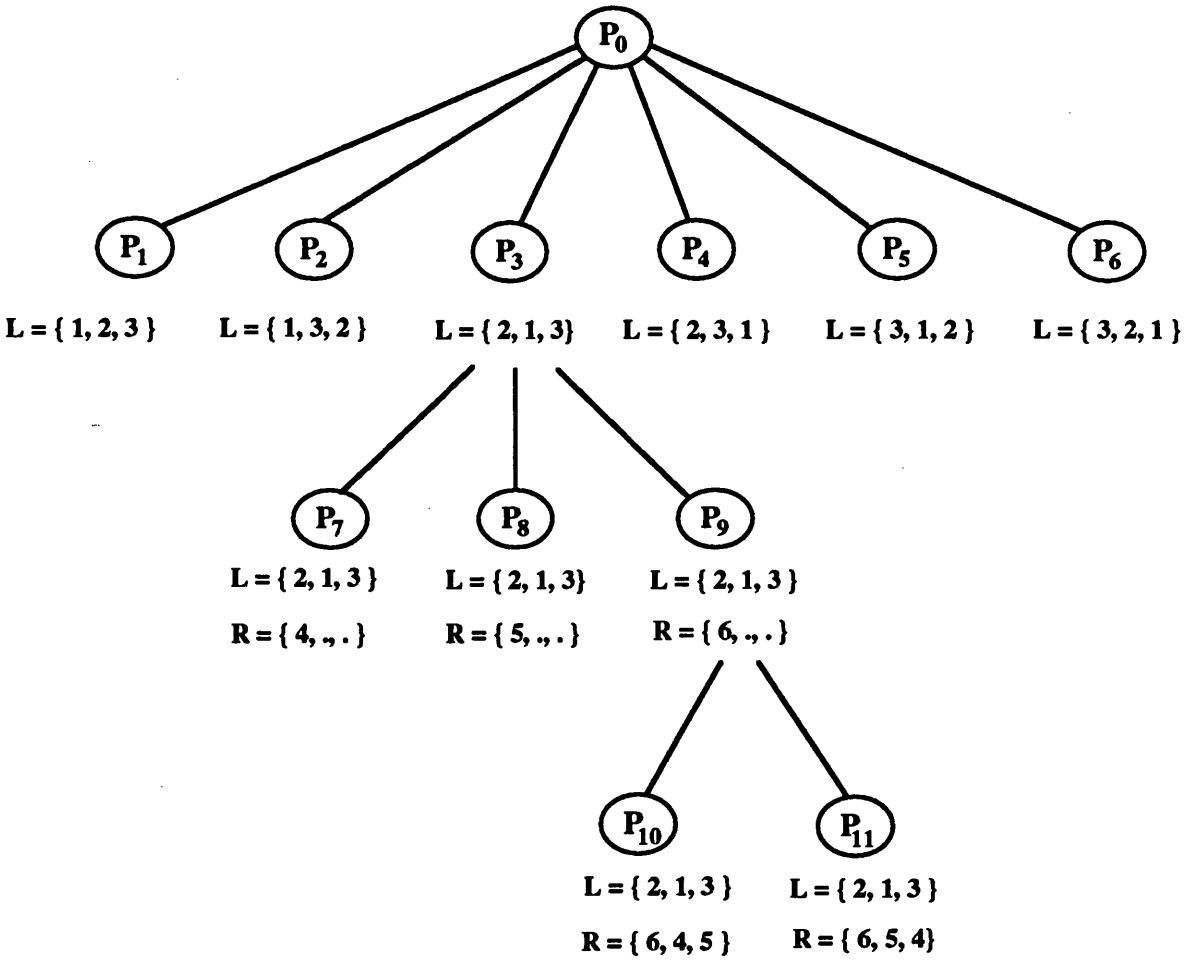


Figura 5. Ejemplo del árbol de soluciones

2.2 COTAS INFERIORES DE UN NODO.

2.2.1 NODOS DEL PRIMER NIVEL DE RAMIFICACION.

PRIMERA COTA INFERIOR

En cada nodo de la primera ramificación (en los que se han fijado todos los vértices del lado izquierdo) se puede obtener una cota inferior mediante el siguiente procedimiento:

Se construye un grafo auxiliar H dirigido y completo con costes en las aristas. Los vértices en dicho grafo son los vértices de R y para cada par de vértices i,j, existe la arista (i,j) y la (j,i). El peso de la arista (i,j) en dicho grafo auxiliar es K(i,j), tomando como ordenación del lado izquierdo la fijada en el nodo.

Ejemplo: Sea el grafo de la figura 6 y consideremos el nodo en el que la ordenación del lado izquierdo es {1,2,3}. En dicho nodo se tiene el grafo H que muestra la figura 7.

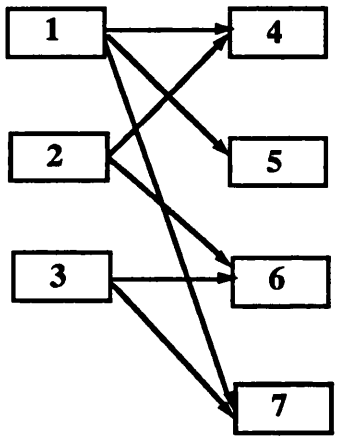


Figura 6. Grafo original

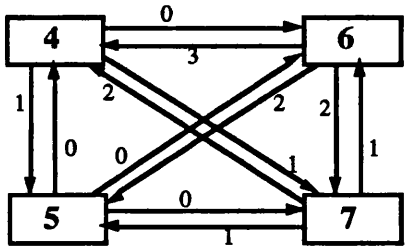


Figura 7. Grafo auxiliar H

Es equivalente hablar de ordenaciones del lado R del grafo original que de caminos hamiltonianos del grafo auxiliar H. Además, dado un camino hamiltoniano $v_{i_1}, v_{i_2}, ..., v_{i_n}$, el coste del camino es:

$$K(v_{i_1}, v_{i_2}) + K(v_{i_2}, v_{i_3}) + ...+ K(v_{i_{n-1}}, v_{i_n})$$

Es decir: la suma del número de intersecciones de todos los pares de aristas incidentes con vértices contiguos en el camino, considerando como ordenación del nivel

derecho la proporcionada por el camino. Mientras que el número total de intersecciones al considerar la ordenación proporcionada por el camino como ordenación de R es:

$$\sum_{j=1}^{n-1} \sum_{k>j} K(v_{i_j}, v_{i_k})$$

La diferencia entre ambas expresiones estriba en que, en la primera no se evalúan las intersecciones debidas a aristas incidentes con vértices no contiguos en el camino.

Así, en el ejemplo anterior un camino hamiltoniano es el 5,4,6,7 con coste 2 y, el grafo con dicha ordenación (y con la ordenación del lado izquierdo fijada en el nodo) tiene tres intersecciones como muestra la figura 8.

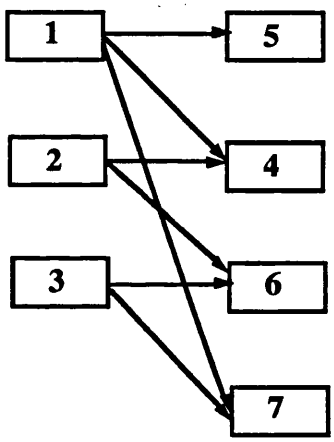


Figura 8

Ello es debido a que en el coste del camino sólo se ha evaluado K(5,4)+K(4,6)+K(6,7)=2, pero no se ha considerado K(4,7), que en este caso vale 1. Es decir; en el peso de dicho camino se evalúa el número de cortes de aristas que llegan al vértice 5 con aristas que llegan al 4, igualmente, de aristas que llegan al 4 con aristas que llegan al 6 y de aristas que llegan al 6 con aristas que llegan al 7; pero no se considera el número de intersecciones debidas a otras parejas de vértices, como por ejemplo, los vértices 4 y 7 que en este ejemplo producen una intersección.

El peso del **camino hamiltoniano de mínimo peso** es menor o igual que el número de intersecciones del grafo original para cualquier ordenación del lado derecho, considerando la ordenación izquierda la fijada en el nodo. Por ello, dicho peso es una cota inferior del nodo. En adelante llamaremos a dicha cota L_1.

SEGUNDA COTA INFERIOR

En [9], Eades y Kelly definen el óptimo derecho como la ordenación del lado derecho que proporciona el menor número de intersecciones de aristas para una ordenación fija del lado izquierdo. Para dicho óptimo derecho proponen una cota inferior que les permite medir la bondad de los algoritmos heurísticos diseñados. Dicha cota inferior viene dada por la siguiente expresión:

Sea l una ordenación fija del lado izquierdo, para cualquier ordenación del lado derecho, el número de intersecciones K(G) del grafo es superior a:

$$\frac{1}{2} \sum_{i,j \in R} \min(K(i,j) , K(j,i))$$

Esta cota inferior, se basa en el hecho de que, dados dos vértices i,j, en cualquier ordenación i precede a j o j precede a i. Luego en cualquier ordenación, el número de intersecciones de aristas incidentes con i y j, está acotado inferiormente por el mínimo de K(i,j) y K(j,i). El número total de intersecciones K(G) generadas por una ordenación se calcula mediante la suma de todos los K(i,j) para todas las parejas i,j en donde i precede a j, como se mostró en la expresión (1) del punto 1.2,. Los dos argumentos demuestran que la expresión dada es una cota inferior al número de intersecciones del grafo[1].

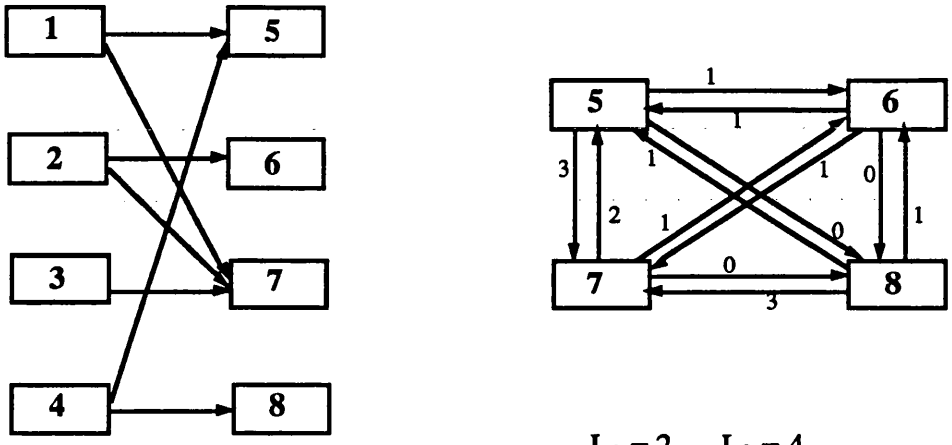
En los nodos de la primera ramificación que estamos estudiando, el orden del lado izquierdo es fijo, con lo cual se puede considerar dicha cota inferior como la cota inferior de un nodo. Una vez construido el grafo auxiliar H, la cota inferior propuesta por Eades y Kelly equivale a, para cada par de aristas que unen dos vértices, tomar la de coste menor (si hay empate, tomar una cualquiera de las dos) y después sumar dichos costes para todos los pares de vértices del grafo auxiliar. En adelante llamaremos a dicha cota L_2.

En el ejemplo mostrado en las figuras 6 y 7, esta expresión produce una cota inferior igual a 2.

[1]Notar que el coeficiente de 1/2 que aparece en la expresión, se utiliza para que en la suma final no aparezca repetido dos veces el mismo sumando (uno asociado al par i,j y otro al j,i).

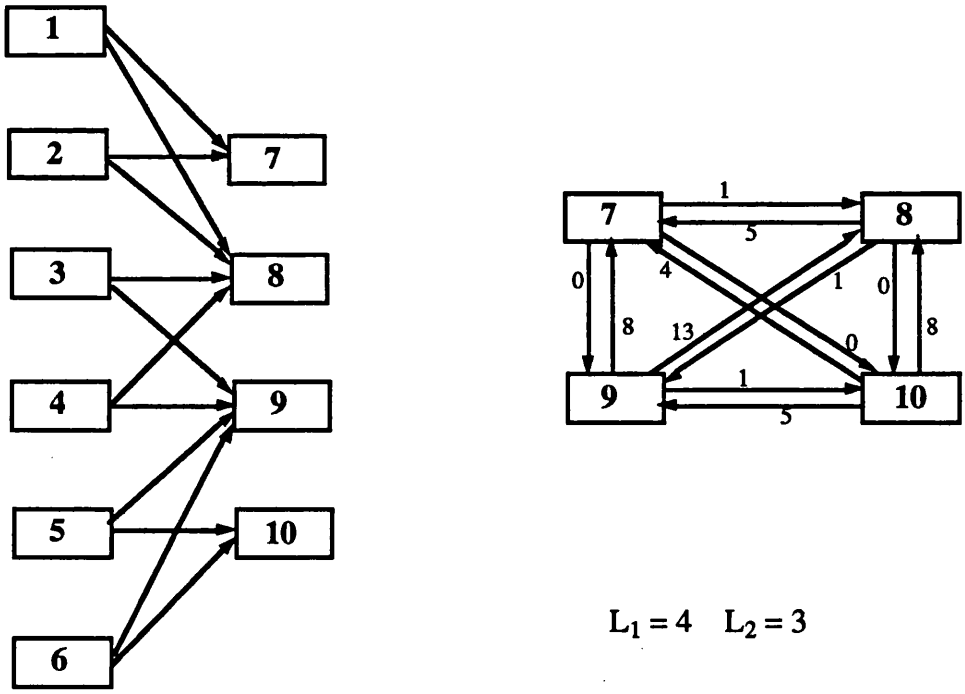
COMPARACION DE LAS DOS COTAS.

Ninguna de las dos cotas domina a la otra, como muestran los siguientes ejemplos:



$L_1 = 2 \quad L_2 = 4$

Figura 9



$L_1 = 4 \quad L_2 = 3$

Figura 10

Dado que la cota L_2 requiere de un esfuerzo computacional menor que la cota L_1, es conveniente calcular primero ésta, ya que si fuera menor que la cota superior global, permitiría saturar el nodo sin necesidad de calcular la cota L_1.

2.2.2 NODOS DE UN NIVEL DE RAMIFICACION MAYOR QUE UNO

Consideremos un nodo P del nivel de ramificación i con $i \geq 2$. Todas las soluciones representadas por P tienen la misma ordenación del lado izquierdo (la ordenación determinada por el único nodo del primer nivel del que P es sucesor), y tienen fijados los i-1 primeros vértices de la ordenación del nivel derecho.

Para cada solución del nodo P, se puede obtener el número de intersecciones K mediante la suma de tres parámetros: $K= k_1 + k_2 + k_3$, donde k_1 es el número de intersecciones producidas por aristas incidentes con los vértices ya fijados, k_2 el número de intersecciones producidas por las parejas de aristas en las que una es incidente con un vértice fijado y la otra con uno no fijado y, k_3 es el número de intersecciones de aristas incidentes con los vértices no fijados en el nodo.

Notar que los parámetros k_1 y k_2 son invariantes para cualquier solución del nodo P. Obviamente k_3 depende de cómo se ordenen los vértices de R no fijados en P.

Ejemplo:

Consideremos el grafo de la figura 11. Sea P el nodo en el que el orden de los vértices del lado izquierdo es el {1,2,3} y en el lado derecho se han fijado los vértices 7,5 y 9 en ese orden. Los vértices 4, 6 y 8 no han sido fijados. En dicho nodo se tiene que:

$k_1 = k(7,5)+k(5,9)+k(7,9) = 0+3+1 = 4$

$k_2 = k(7,4)+k(7,6)+k(7,8) + k(5,4)+k(5,6)+k(5,8) + k(9,4)+k(9,6)+k(9,8) =1+ 0+ 1+ 2+ 1+ 2+ 1+ 0+ 1 = 9$

El valor de k_3 depende del orden de los vértices {4,6,8} en la solución concreta que se estudie

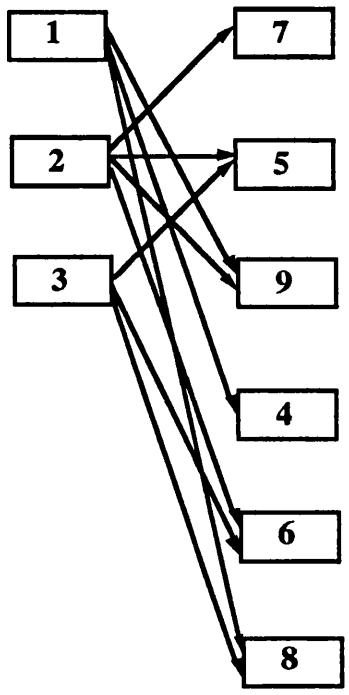


Figura 11

PRIMERA COTA INFERIOR

De manera análoga a los nodos del primer nivel de ramificación, se construye el grafo auxiliar H con los vértices no fijados en el nodo. Sea k_4 el peso del camino hamiltoniano de mínimo peso del grafo H. k_4 es una cota inferior de k_3. De este modo se define una cota inferior al número de intersecciones de las soluciones del nodo como: $L_3 = k_1 + k_2 + k_4$

Ejemplo:

Considerando el ejemplo anterior con el grafo de la figura 11, se tiene que, en dicho nodo el grafo auxiliar H es el que muestra la figura 12. El camino hamiltoniano de mínimo peso es el 5,7,6. El peso de dicho camino es $k_4 = 1$, por lo que, la cota inferior L_3 es:

$$L_3 = 4+9+1=14.$$

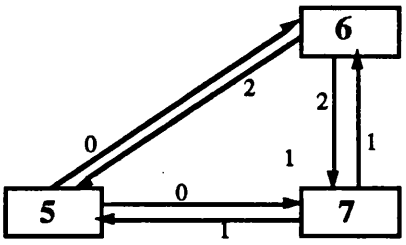


Figura 12. Grafo H

SEGUNDA COTA INFERIOR

Al igual que en los nodos del primer nivel de ramificación, sea k_5 el peso del grafo que se obtiene del grafo H asociado a los vértices no fijados, eliminando de cada par de aristas (i,j) (j,i) la de mayor peso (en caso de empate, eliminar una cualquiera de las dos). Entonces $L_4 = k_1 + k_2 + k_5$ es una cota inferior del nodo P.

En el ejemplo estudiado se obtiene un valor de $k_5 = 1$, como se puede ver en la figura 12, por lo que la cota $L_4 = 4+9+1=14$.

Así pues, para cada nodo se considerarán las dos cotas inferiores propuestas: L_3 y L_4, tomando la mayor de ambas como la cota inferior del nodo.

CALCULO DEL CAMINO HAMILTONIANO DE MINIMO PESO.

El camino hamiltoniano de mínimo peso del grafo auxiliar se puede obtener hallando un s-t flujo de coste mínimo y valor 1 en la red T definida de la siguiente manera:

Vértices:	Todos los del grafo auxiliar y además dos ficticios: s y t.
Aristas:	Todas las del grafo auxiliar y, además, dos por cada vértice:una desde s al vértice y otra desde el vértice a t.
Costes:	Las aristas del grafo auxiliar mantienen su coste, las añadidas tienen un coste 0.
Capacidad:	La capacidad de todas las aristas es de uno.
Cotas :	Todos los vértices del grafo auxiliar tienen cota inferior y superior igual a uno.
Oferta:	El vértice s tiene oferta uno.
Demanda:	El vértice t, tiene demanda uno.

El algoritmo, al calcular el flujo, seguirá un camino que recorrerá todos los vértices una sola vez, ya que tienen cota inferior y superior igual a uno, con lo cual se define un camino hamiltoniano. Como el flujo es de coste mínimo, dicho camino será de mínimo peso. Veamos en el ejemplo de la figura 7 como funciona este primer método.

A partir del grafo auxiliar se construye el grafo T:

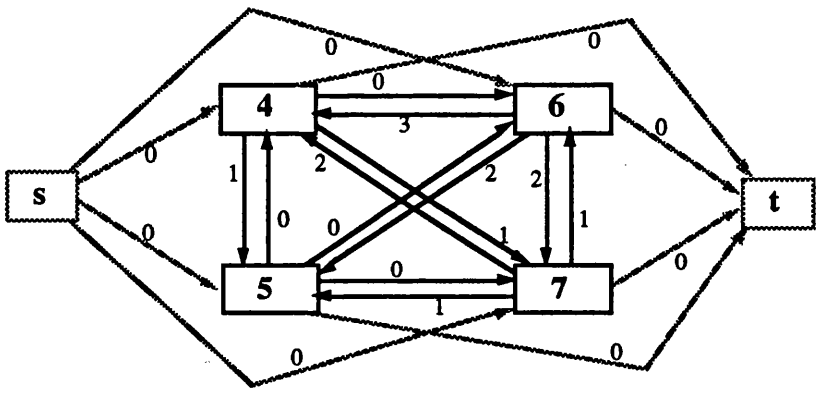


Figura 13. Grafo T

en donde un s-t flujo de coste mínimo y valor uno es: s - 5 - 7 - 6 - 4 - t, que tiene asociado un coste de dos, por lo que un camino hamiltoniano de mínimo peso en el grafo auxiliar es el 5,7,6,4.

2.3 MEJOR SOLUCION ASOCIADA A UN NODO.

Veamos un procedimiento que permitirá obtener directamente, en algunos casos, la mejor solución asociada a un nodo.

Consideremos un nodo cualquiera del árbol, bien de la primera ramificación o de cualquier otra y, sea H el grafo auxiliar dirigido y completo asociado a dicho nodo. A partir de H se construye el grafo H* sobre el mismo conjunto de vértices que H de la siguiente manera. Por cada par de aristas que conectan dos vértices i,j en H, si tienen pesos diferentes (K(i,j)<K(j,i)), se sitúa en H* la arista dirigida de menor peso. Si ambas tienen el mismo peso (K(i,j)=K(j,i)), se sitúa en H* una arista no dirigida con peso K(i,j), uniendo los dos vértices. El grafo resultante H* es mixto (tiene aristas dirigidas y no dirigidas) y cumple que, el grado de cualquier vértice es n-1 (considerando $d(v)=d_e(v) + d_s(v)$). Llamaremos a H* subgrafo simple mínimo de H.

Si el nodo es del primer nivel de ramificación, sumando los pesos de todas las aristas de H* se obtiene la cota inferior L_2. Si el nodo es de un nivel de ramificación mayor, a la suma de dichos costes hay que añadirle k_1 y k_2 para obtener la cota inferior L_4.

Los siguientes resultados permitirán encontrar, a partir de H* y bajo ciertas condiciones, la mejor solución del nodo.

TEOREMA 2

Sea G un grafo dirigido, simple, con n vértices, y tal que $d(v)=n-1\ \forall v$. Entonces:

G es acíclico si y sólo si existe un único camino hamiltoniano en G.

Prueba

[←] Sea $v_1, v_2, ...,v_n$ el único camino hamiltoniano en G.

Vamos a demostrar que $d_e(v_i) = i-1$ (y por lo tanto $d_s(v_i) = n-i$).

Puesto que $d(v_1) = n-1$, v_1 es incidente con el resto de vértices. Así pues, $\forall k>1$ existe (v_1,v_k) o (v_k,v_1) pero no ambas.

Si $d_e(v_1) \neq 0$, sea v_k el último vértice del camino hamiltoniano tal que existe (v_k,v_1). Entonces $v_2, v_3, ...,v_k, v_1, v_{k+1}, ...,v_n$, si $k \neq n$, o $v_k, v_1, v_2, ...,v_{n-1}$, si k=n, es otro camino hamiltoniano, lo cual es absurdo, luego $d_e(v_1) = 0$.

Sea $G_1 = G - \{v_1\}$. Entonces $v_2, ...,v_n$ es el único camino hamiltoniano en G_1, el número de vértices de G_1 es n-1 y se cumple que $d^{G1}(v_i)=n-2$, i=2, ..,n.

Luego $d_e^{G1}(v_2) = 0$, por lo que $d_e^{G}(v_2) = 1$.

El mismo razonamiento sobre los grafos $G_i = G_{i-1} - \{v_i\}$, i=2, .., n-1 demuestra que $d_e(v_i) = i-1$, i=3, ..,n.

Dado que v_1 tiene grado de entrada cero, no puede pertenecer a ningún circuito. Considerando que v_2 tiene grado de entrada uno y, que la arista que le llega parte de v_1, v_2 no puede pertenecer a ningún circuito. Análogamente v_3 tiene grado de entrada dos y necesariamente le llega una arista de v_1 y otra de v_2, por lo que no puede pertenecer a ningún circuito. De esta manera, en n pasos se prueba el resultado.

[→] Por inducción sobre n. Para n=1 el resultado es trivialmente cierto.

Supongamos que es cierto para n y demostremos que es cierto para n+1.

Sea G un grafo acíclico con n+1 vértices y tal que $d(v)=n \ \forall v$. Por ser G acíclico, existe un vértice v / $d_e(v)= 0$ luego, $d_s(v)= n$. Entonces G-{v} es un grafo de n vértices que cumple las condiciones del teorema. Por hipótesis de inducción, contiene un único camino hamiltoniano P. Entonces {v,P} es el único camino hamiltoniano de G. ■

COROLARIO 1

Sea G un grafo dirigido, simple, con n vértices, y tal que $d(v)=n-1 \ \forall v$. Entonces:

G es acíclico si, y sólo si, existe una ordenación $(v_1, v_2, ...,v_n)$ de los vértices de G, tal que $d_e(v_i) = i-1$, i=1, ..,n.

Prueba

[→] A partir de G acíclico, aplicando el teorema 2, existe un único camino hamiltoniano $\{v_1, v_2, ...,v_n\}$ en G, donde necesariamente $d_e(v_i) = i-1$, i=1, ..,n.

[←] Como se vio en la demostración del teorema 2, $v_1, v_2, ...,v_n$, no pueden pertenecer a ningún circuito.■

TEOREMA 3

Sea H el grafo auxiliar de un nodo del árbol de soluciones. Si $K(i,j) \neq K(j,i)$ $\forall i,j \in H$ y H* es acíclico, entonces:

El único camino hamiltoniano de H* proporciona la solución óptima del nodo.

Prueba

Supongamos que estamos en un nodo del primer nivel de ramificación. Por el teorema 2 y por el corolario 1, H* tiene un único camino hamiltoniano $P=\{v_1, v_2, ...,v_n\}$ y los vértices de P cumplen que $d_e(v_i) = i-1$, $i=1, ..,n$. Ello implica que $\cup_{i=1, ..,n} \{ (v_i, v_j) / j>i \}$ es el conjunto de aristas de H* y que, por lo tanto, el peso de H* es:

$$p(H^*) = \sum_{i=1}^{n-1} \sum_{j>i} k(v_i, v_j) \qquad (1)$$

es decir, el número de intersecciones de las aristas del grafo G cuando la ordenación de R es $(v_1, v_2, ...,v_n)$. P es, también, un camino hamiltoniano en H, y, por lo tanto, una ordenación de R. Veamos que es la que proporciona el menor número de intersecciones.

Sea $(v_{i_1},v_{i_2}, ...,v_{i_n})$ una ordenación cualquiera de R. El número de intersecciones asociado es:

$$\sum_{j=1}^{n-1} \sum_{k>j} k(v_{ij},v_{ik}) \qquad (2)$$

Las expresiones (1) y (2) tienen el mismo número de sumandos, uno por cada par de vértices de R. Sin embargo , por construcción, para cada par de vértices de R, el sumando correspondiente en (1) es menor o igual que su correspondiente sumando en (2). Así pues, P proporciona la solución óptima del nodo.

Consideremos ahora un nodo en un nivel de ramificación $k>1$. Entonces se han fijado k-1 vértices de la ordenación de R. Supongamos que son $(v_1, v_2, ...,v_{k-1})$ por este orden.

Por el mismo razonamiento que antes, la ordenación proporcionada por P es la que produce el menor número de intersecciones entre aristas incidentes con los n-k+1 vértices

no fijados en el nodo. Por lo tanto (v_1, v_2, ...,v_{k-1}, P) es la ordenación óptima del nodo. ■

Ejemplo:

Consideremos el grafo de la figura 14. Sea un nodo de la primera ramificación en donde el orden del lado izquierdo es el 1,2,3. Con el procedimiento descrito se construye el grafo H de este nodo (figura 15) y, a partir de éste, el grafo H* (figura 16).

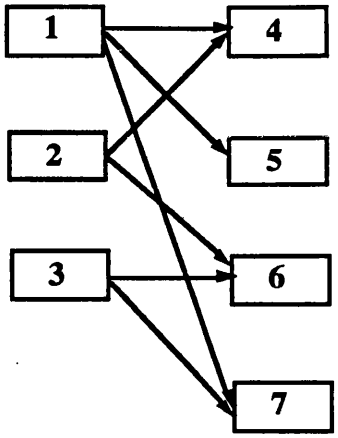


Figura 14. Grafo original

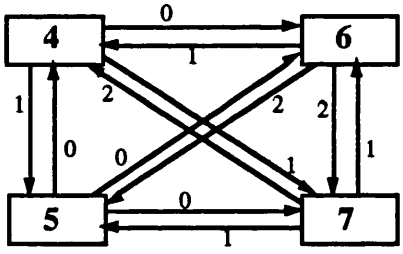


Figura 15. Grafo H

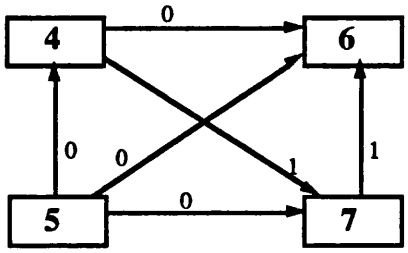


Figura 16. Grafo H*

Dado que H* es acíclico, el único camino hamiltoniano es el {5,4,7,6} con peso 2. Por lo que la ordenación 5,4,7,6 de los vértices del lado derecho de G produce la solución óptima del nodo.

Notar que el corolario 1 proporciona un método eficiente para saber si H* es acíclico o no.

Definición

Sea H un grafo mixto.

Una orientación de H es un grafo dirigido que se obtiene de H sustituyendo cada arista no dirigida (i,j) de H por, o bien la arista dirigida (i,j), o bien, la arista dirigida (j,i).

TEOREMA 4

Sea H el grafo auxiliar de un nodo del árbol de soluciones y H* su subgrafo simple mínimo. Sea E* el conjunto de las aristas no dirigidas de H*. Entonces,

Si el grafo H* - E* es acíclico, existe una orientación de H* que es acíclica.

Prueba

Vamos a establecer un procedimiento que, iterativamente, va a ir orientando las aristas de E* hasta obtener una orientación acíclica de H*.

Puesto que H*-E* es acíclico, existe un vértice v_1 cuyo grado de entrada en H*-E* es cero. Orientamos, si existen, las aristas no dirigidas incidentes con v_1 de manera que salgan de v_1. Sea H_1* el grafo que se obtiene de H* al orientar las aristas mencionadas y sea E_1* el conjunto de aristas no dirigidas de H_1*. Entonces H_1*-E_1* es acíclico, puesto que H*-E* era acíclico y las aristas que se han añadido a H*-E* para obtener H_1*-E_1* salen todas de v_1.

Como H_1*-E_1* es acíclico, H_1*-E_1*-{v_1} es también acíclico. Por lo tanto, existe en H_1*-E_1*-{v_1} un vértice v_2 cuyo grado de entrada en él es cero. Se orientan, si existen, las aristas no dirigidas incidentes con v_2 de manera que salgan de v_2.

Sea H_2* el grafo que se obtiene de H_1* al orientar las aristas mencionadas y sea E_2* el conjunto de aristas no dirigidas de H_2*. Entonces H_2*-E_2* es acíclico, puesto que las aristas que hay que añadir al grafo acíclico H_1*-E_1* para obtener H_2*-E_2* salen todas de v_2, y, en H_2*-E_2* la única arista que llega a v_2 sale de v_1 que tiene grado de entrada cero.

El procedimiento se continúa hasta que todas las aristas de E* han sido orientadas. El último H_k* obtenido es una orientación acíclica de H*. ■

COROLARIO 2

Sea H el grafo auxiliar de un nodo del árbol de soluciones y H* su subgrafo simple mínimo. Si H* es mixto,entonces:

Cada orientación acíclica de H* proporciona una solución óptima del nodo.

Prueba

Sea H_1* una orientación acíclica de H*. Por el teorema 2 y por el corolario 1, H_1* tiene un único camino hamiltoniano P={v_1, v_2, ...,v_n} y los vértices de P cumplen que

$d_e(v_i) = i-1$, $i=1, ..,n$. A partir de este camino P y con la misma demostración que la del teorema 3 se prueba el resultado. ■

Así pues, el procedimiento de construcción de la orientación acíclica de H* descrito en el Teorema 4, proporciona una solución óptima del nodo.

Ejemplo:

Consideremos el grafo de la figura 17 y sea un nodo del primer nivel de ramificación, donde el orden del lado izquierdo es el 1,2,3,4. A partir del grafo H (figura 18) se construye el grafo mixto H* (figura 19).

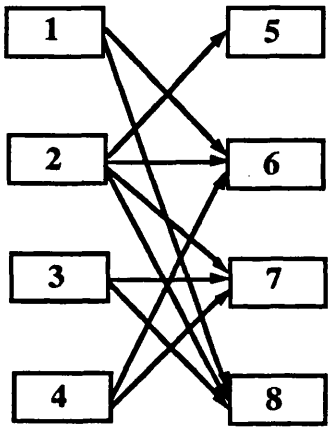


Figura 17. Grafo original

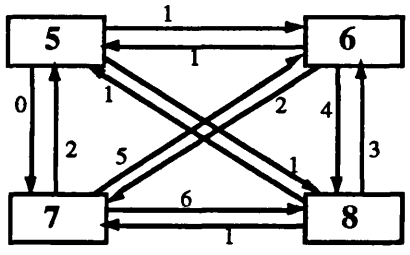


Figura 18. Grafo H

El conjunto de aristas no dirigidas de H* es E*={ (5,6) (5,8)}. Dado que H*-E* es acíclico, aplicando el teorema 4, se obtiene una orientación acíclica de H* (figura 20).

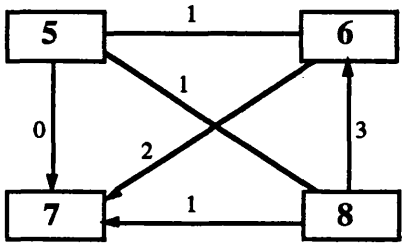


Figura 19. Grafo mixto H*

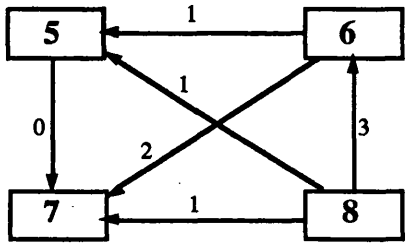


Figura 20. Orientación acíclica de H*

Dado que la orientación de H* es acíclica, el único camino hamiltoniano es el {8,6,5,7} con peso 4. Por lo que, la ordenación 8,6,5,7 de los vértices del lado derecho de G, proporciona una solución óptima del nodo.

El siguiente ejemplo muestra el caso de un nodo en el que H* no es acíclico.

Consideremos un nodo en el que el grafo auxiliar H es el de la figura 21. Se tiene que:

k(9,10)=10 < k(10,9) =9
k(9,11) =8 < k(11,9) =9
k(11,10)=6 < k(10,11)=8

Por lo que, el grafo H* será el que muestra la figura 22, que contiene un circuito.

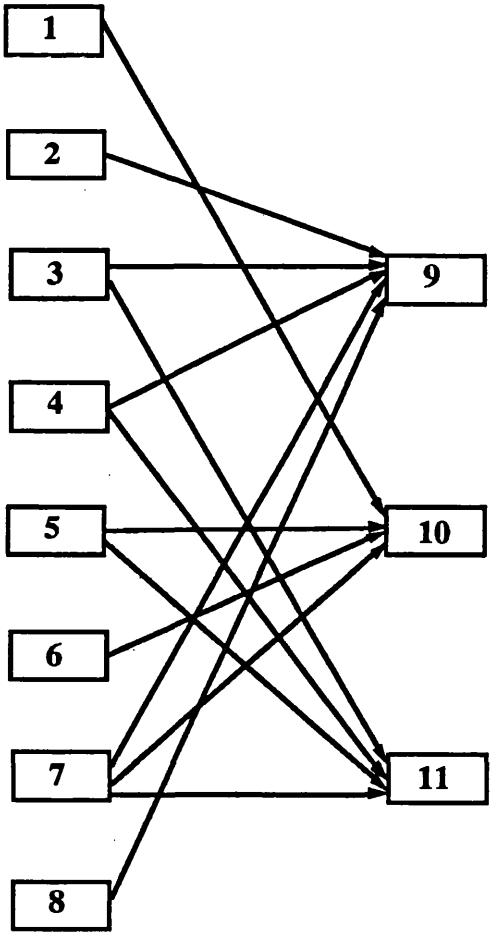


Figura 21

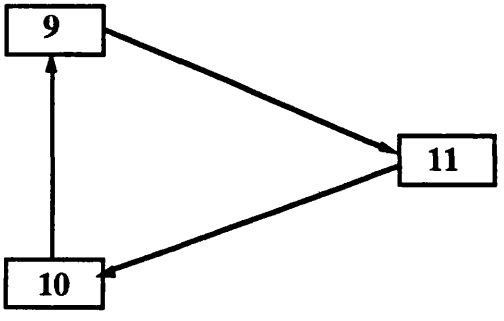


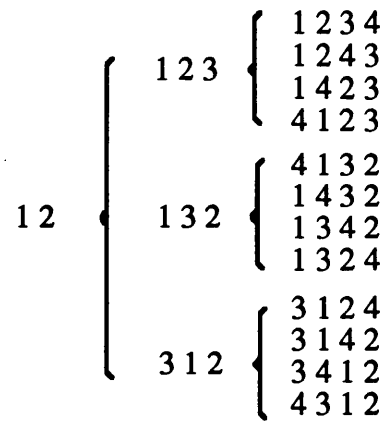
Figura 22

2.4 ESTRATEGIA DE EXPLORACION

2.4.1 ARBOL DE SOLUCIONES AMPLIADO

En [38], E.M. Reingold y otros, proponen un algoritmo para generar las n! permutaciones de n elementos, de manera que, dos permutaciones sucesivas sólo difieran en el intercambio de dos elementos consecutivos. El algoritmo construye las n! permutaciones de los elementos {1,2, ..,n} a partir de las (n-1)! permutaciones de los elementos {1,2, .., n-1}, insertando el elemento n en todas las posiciones posibles.

El diagrama de la derecha muestra la secuencia de permutaciones generada por el algoritmo mencionado para el conjunto {1,2,3,4}. En el ejemplo sólo se muestran las doce primeras. Las permutaciones han sido generadas a partir de tres de las seis permutaciones del conjunto {1,2,3}.



El algoritmo se basa en realizar la inserción del elemento n en posiciones sucesivas, comenzando por atrás en los grupos impares y por delante en los pares. Así, para cada una de las dos permutaciones de los elementos {1,2}, que son 1,2 y 2,1, se obtienen tres permutaciones insertando el elemento 3. En el ejemplo sólo aparecen la 123, 132, 312 que son las correspondientes al primer grupo. De forma análoga, a partir de éstas se generan las de cuatro elementos.

Considerando el modo en que el algoritmo de Reingold genera las permutaciones, se pueden establecer grupos de permutaciones. Todas las permutaciones de n elementos {1,2, ..,n} que tienen la misma ordenación de los n-1 elementos {1,2, .., n-1} se pueden agrupar en una clase. Dicha clase está representada por la permutación de los n-1 elementos. De este modo, las n! permutaciones de n elementos se agrupan en (n-1)! clases.

Aplicando la misma idea a las clases obtenidas, se pueden agrupas éstas en clases superiores. Así, todas las clases de n-1 elementos que tienen la misma ordenación de los n-2 primeros elementos se pueden agrupar en una clase superior. Dicha clase está representada por la permutación de los n-2 elementos. De este modo, se puede definir de forma recursiva hasta alcanzar las clases de dos elementos. En el árbol de soluciones, cada nodo del primer nivel de ramificación viene definido por una de las permutaciones

de los vértices del lado izquierdo del grafo. A partir de dichas permutaciones se pueden agrupar los nodos en clases.

Definición

Sea G=(L,R,E) con |L|=n y sea un orden inicial en L: L={v_1, v_2, ...,v_n}.

Sean l^i_1, l^i_2, .., $l^i_{i!}$, las permutaciones de los i primeros elementos de L (1<i<n).

Para cada una de dichas permutaciones se define una clase Q de orden i, que representa al conjunto de soluciones posibles en las que la ordenación de los i primeros elementos de L es l^i_j.

La introducción de estas clases amplía el árbol de soluciones de la siguiente manera: Considerando cada una de las clases como un nuevo nodo del árbol de soluciones, se tiene que: A partir del nodo inicial se ramifica en dos clases de orden 2 (una con la ordenación (v_1, v_2) y otra con la (v_2, v_1)). Cada una de las clases de orden 2 se ramifica en tres clases de orden 3. De este modo hasta llegar a las clases de orden n-1, cuyas ramificaciones son los nodos del primer nivel de ramificación.

Considerando el grafo G=(L,R,E) en donde L={1,2,3,4}, el siguiente dibujo muestra una representación parcial del árbol de soluciones ampliado.

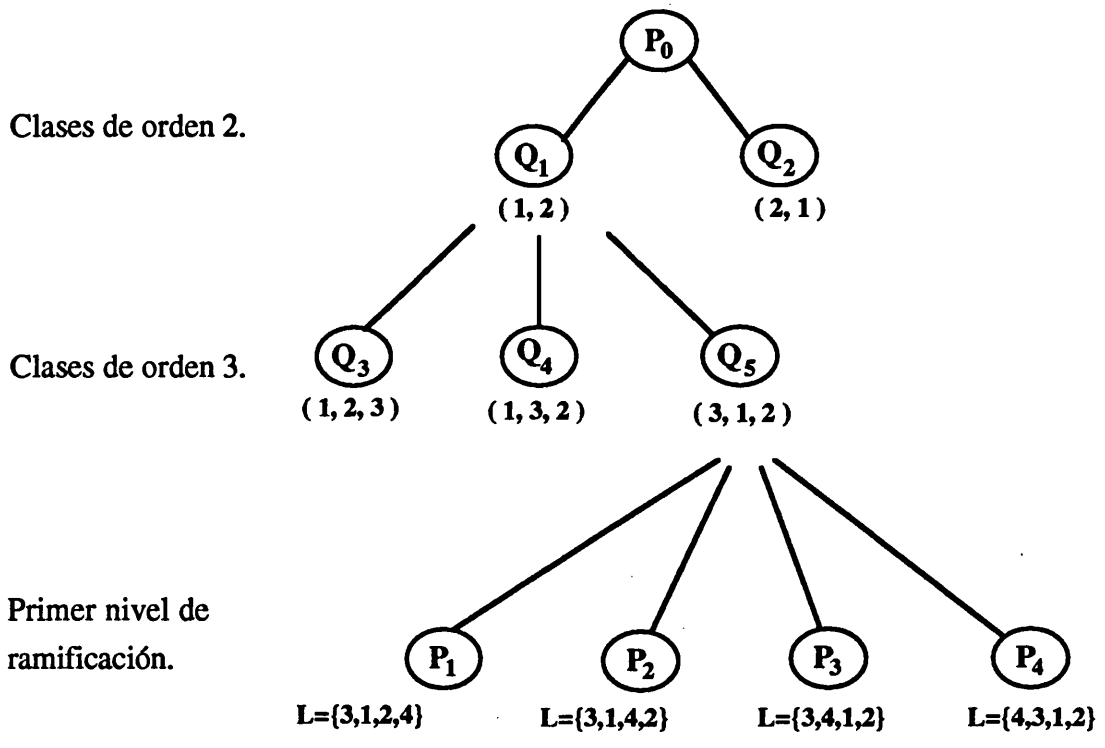


Figura 23. Arbol de soluciones ampliado

2.4.2 COTA INFERIOR DE UNA CLASE DE ORDEN K.

Esta agrupación de los nodos del primer nivel de ramificación en clases, permite obtener una cota inferior de la clase de nodos que, de ser mayor que la superior global, permitiría saturar todos los nodos de la clase.

Consideremos que estamos ramificando según las ordenaciones del lado izquierdo. Si |L|=n, sea G´el grafo resultante de eliminar el vértice n del lado izquierdo y sus aristas incidentes. Cada una de las clases representa una ordenación de todos los vértices del lado izquierdo de G´. Luego se pueden calcular los dos tipos de cotas inferiores de G´ para cualquier ordenación de los vértices del lado derecho. Dichas cotas inferiores son válidas en todos los nodos de la clase.

En orden a que las cotas de cada clase sean lo mejores posibles, se renumeran los vértices del lado izquierdo de G, en orden decreciente de su grado.

Ejemplo:

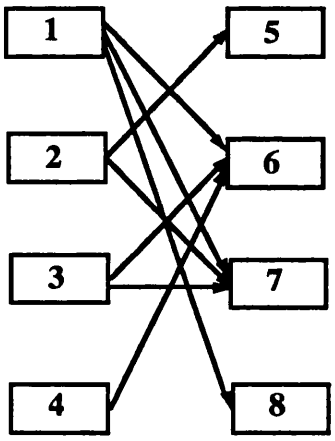


Figura 24. Grafo original

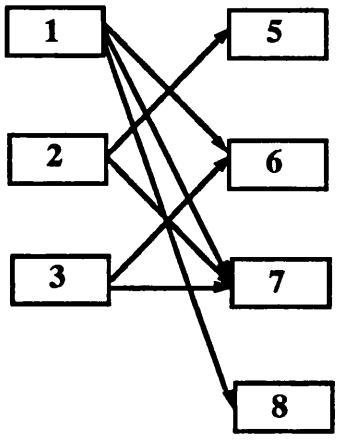


Figura 25. Grafo en clase 1

Como muestran las figuras, a partir del grafo original, se elimina el vértice 4 y sus aristas incidentes (por ser el de menor grado). Fijando el orden izquierdo del grafo resultante a {1,2,3}, se obtiene el grafo que utilizaremos en la clase 1 de los nodos de la primera ramificación para obtener la cota inferior. Una cota inferior de las intersecciones del grafo reducido (sin el vértice 4), lo será de cualquier ordenación del lado izquierdo de G en la que el 1 preceda al 2 y, el 2 al 3.

A partir del grafo de la figura 25 se construye el grafo auxiliar H.

En dicho grafo un camino hamiltoniano de mínimo peso es el 7,5,8,6 con peso 3. Luego la cota inferior L_1 es 3 y la L_2 es 6. Considerando que la ordenación óptima del grafo de la figura 24 tiene dos intersecciones, una cota superior global razonable, permitiría saturar la clase de nodos. Lo cual equivale a saturar los cuatro nodos del primer nivel de ramificación de dicha clase.

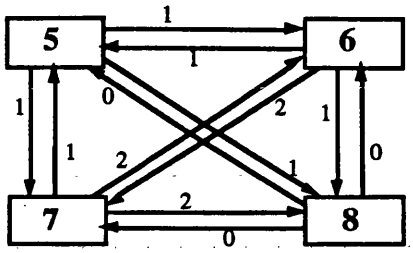


Figura 26

2.4.3 ESTRATEGIA DE EXPLORACION

El árbol de soluciones ampliado se explora mediante la siguiente estrategia D.F.S. (Depth First Search) en donde las soluciones posibles del problema se alcanzan lo antes posible.

Los nodos correspondientes a las clases se examinan "siguiendo la generación de permutaciones del algoritmo de Reingold".

Los nodos de un nivel de ramificación mayor o igual que uno, se examinan en el orden siguiente. Sea $\{v_1, v_2,..,v_m\}$ el orden de los vértices de R en la solución inicial. Sea un nodo P_i del nivel de ramificación $k > 1$, en el que, los vértices del lado izquierdo están todos fijados y en el derecho se han fijado $\{v_{j(1)}, v_{j(2)},..,v_{j(k-1)}\}$.

El nodo P_i se ramifica en m-k+1 nodos hijos, en donde, en cada uno, se fija en la posición k uno de los m-k+1 vértices de R que quedan por asignar. Los nodos hijos se ordenan considerando el orden del vértice que se fija en el nodo, en la solución inicial de R.

Una vez examinado P_i, si no se satura, se continúa la exploración con el primer hijo. En caso de saturarse, se continúa con su nodo hermano siguiente en el orden establecido.

2.5 MEJORAS EN EL ALGORITMO.

2.5.1 NUMERO DE NODOS EN EL PRIMER NIVEL DE RAMIFICACION

Como se comentó al describir el algoritmo, en el primer nivel de ramificación existen tantos nodos como ordenaciones del lado izquierdo del grafo. Esto supone que en un grafo de por ejemplo 10 vértices con 5 a cada lado, se tenga en la primera ramificación 120 nodos. Sin embargo, vamos a ver que basta con considerar la mitad de los nodos.

En la siguiente figura se puede ver como para cada ordenación, su opuesta (desde el final al comienzo) produce un grafo simétrico con el mismo número de intersecciones.

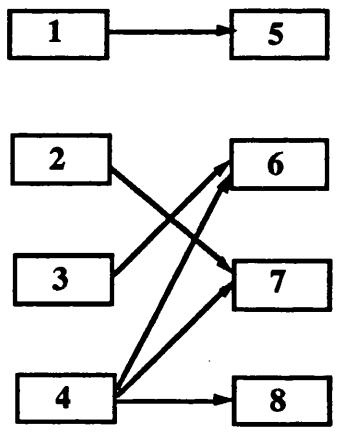


Figura 27.

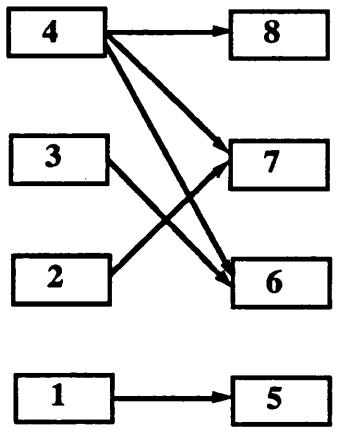


Figura 28

Dado que el objetivo es encontrar una, y sólo una, solución óptima, bastará con enumerar las ordenaciones de manera que ninguna sea la opuesta de otra. Así, al ramificar desde el nodo inicial del árbol se consideran, únicamente, la mitad de las ordenaciones posibles del lado izquierdo. De este modo, al explorar el árbol de soluciones, basta con considerar el subárbol de los nodos descendientes de uno sólo de los dos nodos de las clases de orden 1. Así, en el ejemplo de la figura 23 sólo se examinarían los nodos descendientes de Q_1.

El algoritmo de Reingold genera primero las n! / 2 permutaciones de n elementos en donde ninguna es la opuesta de otra, para generar después las opuestas de todas las anteriores. Así, para explorar el árbol de soluciones de manera que, una vez examinada una solución no se considere su opuesta, basta con realizar la exploración sobre la primera mitad de las clases que genera el algoritmo de Reingold.

2.5.2 CONSTRUCCION EFICIENTE DEL GRAFO H

En este apartado vamos a estudiar como obtener el grafo auxiliar de un nodo a partir del grafo auxiliar del nodo padre o del nodo hermano. De este modo, al explorar el árbol de soluciones, una vez construido el grafo auxiliar H de un nodo, se obtendrán los grafos auxiliares H de los nodos que se examinen a continuación, con el menor esfuerzo computacional posible.

DE UN NODO AL NODO HERMANO

TEOREMA 5

Sea G=(L,R,E) un grafo bipartido y |L|=n.

Sea P_1 un nodo del primer nivel de ramificación en el que el orden de los vértices de L es $(v_1, v_2,..,v_i,v_{i+1},..,v_n)$. Sea H_1 el grafo auxiliar del nodo y $K_1(u,v)$ el coste de la arista dirigida (u,v) de H_1.

Sea P_2 un nodo del primer nivel de ramificación en el que el orden de los vértices de L es el mismo que el de P_1, salvo por la trasposición v_i, v_{i+1}: $(v_1, v_2,..,v_{i+1},v_i,..,v_n)$. Sea H_2 el grafo auxiliar del nodo y $K_2(u,v)$ el coste de la arista dirigida (u,v) de H_2.

Sean los conjuntos:

$$A=\{ (u,v) \in R \times R / (v_i,u)\in E \ (v_{i+1},v)\in E \}$$
$$B=\{ (u,v) \in R \times R / (v_i,v)\in E \ (v_{i+1},u)\in E \}$$

Entonces se verifica:

$$K_2(u,v) = K_1(u,v) \qquad \forall(u,v) \in A\cap B$$
$$K_2(u,v) = K_1(u,v) + 1 \qquad \forall(u,v) \in A-B$$
$$K_2(u,v) = K_1(u,v) - 1 \qquad \forall(u,v) \in B-A$$
$$K_2(u,v) = K_1(u,v) \qquad \forall(u,v) \in R \times R - A\cup B$$

Prueba

Por definición, los dos grafos tienen el mismo conjunto de vértices R y son dirigidos y completos. Dados dos vértices u,v de R, K(u,v) proporciona el número de intersecciones de las aristas que llegan a u con las aristas que llegan a v, considerando que u precede a v en la ordenación de R. Luego al permutar los vértices v_i y v_{i+1}, solo variarán los parámetros k(u,v) en el caso en que existan las parejas de aristas $\{(v_i,u),(v_{i+1},v)\}$ o $\{(v_i,v),(v_{i+1},u)\}$.

Luego, $K_2(u,v) = K_1(u,v) \ \forall(u,v) \in R \times R - A\cup B$.

Si una pareja (u,v) pertenece a ambos conjuntos ($A \cap B$), entonces existen las cuatro aristas: $\{(v_i,u),(v_{i+1},v),(v_i,v),(v_{i+1},u)\}$, lo cual define un ciclo que siempre producirá una intersección. Por ello, en este caso no varía el valor de k(u,v) de un grafo al otro.

Si una pareja (u,v) pertenece a A-B, entonces existen las aristas $\{(v_i,u),(v_{i+1},v)\}$ y posiblemente (v_i,v) o (v_{i+1},u), pero no ambas. De existir la arista (v_i,v), nunca cortará a (v_i,u) ni a (v_{i+1},v) por compartir un vértice con ambas. Lo mismo ocurre con la arista (v_{i+1},u). Por otro lado, si u precede a v, las aristas $\{(v_i,u),(v_{i+1},v)\}$ no se cortan en H_1 y si lo hacen en H_2, por lo que $K_2(u,v) = K_1(u,v) + 1$ y $K_2(v,u) = K_1(v,u) - 1$. Notar que si $(u,v) \in A$ entonces $(v,u) \in B$, además, por el Teorema 1, $K(u,v)+K(v,u)$ es constante para cualquier ordenación de L. ■

Ejemplo:

Sea el grafo de la figura 29. Sea el nodo Nodo P_1 con el orden L={1,2,3}. Sea el nodo Nodo P_2 con el orden L={1,3,2}.

Entonces A={(4,6),(4,7),(6,7)}
B={(6,4),(7,4),(7,6)}

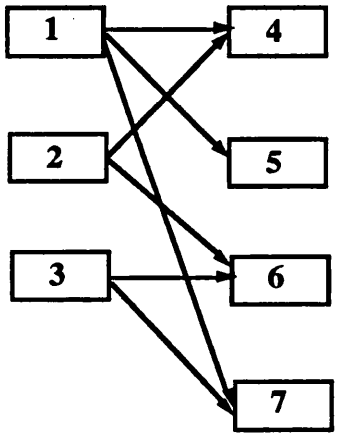


Figura 29

Luego, para construir H_2 (figura 31) a partir de H_1 (figura 30), basta con sumar un uno a los pesos de las aristas asociadas a los elementos de A y restar un uno a los pesos de las asociadas a B.

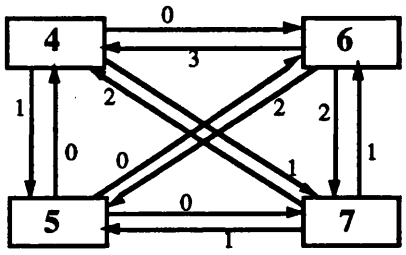


Figura 30. H_1

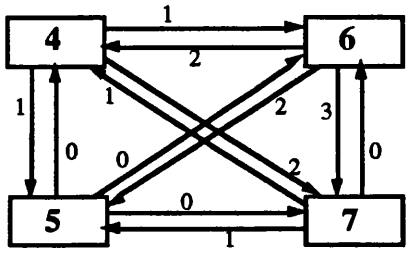


Figura 31. H_2

El teorema 5 es válido exactamente igual para el caso de dos clases del mismo orden que, sólo se diferencian en una permutación en los vértices fijados en el nodo.

Se pueden ver fácilmente los dos teoremas siguientes:

TEOREMA 6

Sea G=(L,R,E) un grafo bipartido.

Sea P_i un nodo del nivel de ramificación $k > 1$ en el árbol de soluciones. Sea H_i el grafo auxiliar del nodo y sean (v_1, v_2,..,v_{k-2}, a) los vértices de R fijados en el nodo.

Sea P_j un nodo hermano de P_i. Sea H_j el grafo auxiliar del nodo P_j y sean (v_1, v_2,..,v_{k-2}, b) los vértices de R fijados en el nodo P_j.

Entonces:

El grafo H_i es igual al grafo H_j salvo por los vértices a y b y las aristas incidentes con ellos.

DEL NODO PADRE AL HIJO

TEOREMA 7

Sea G=(L,R,E) un grafo bipartido.

Sea P_i un nodo del nivel de ramificación i en el árbol de soluciones y sea H_i el grafo auxiliar del nodo.Sea P_{i+1} un nodo hijo de P_i en el que se ha fijado el vértice v_k de R. Sea H_{i+1} el grafo auxiliar del nodo.

Entonces:

El grafo H_i es igual al grafo H_{i+1} menos el vértice v_k y las aristas incidentes.

El teorema siguiente, permite obtener el grafo auxiliar del nodo de una clase, a partir del grafo auxiliar del nodo padre.

TEOREMA 8

Sea G=(L,R,E) un grafo bipartido y |L|=n.

Sea Q_1 una clase de orden i del árbol de soluciones ampliado definida por la permutación (v_1, v_2,..,v_i). Sea H_1 el grafo auxiliar del nodo y $K_1(u,v)$ el coste de la arista dirigida (u,v) de H_1.

Sea Q_2 la clase de un nodo hijo de Q_1 definida por la permutación $(v_1,v_2,..,v_k,b,v_{k+1},..,v_i)$. Sea H_2 el grafo auxiliar del nodo y $K_2(u,v)$ el coste de la arista dirigida (u,v) de H_2. Sea $A \subseteq R$ el conjunto de vértices adyacentes al vértice b.

Para cada vértice v de R se definen: $d_1(v) = | \{(v_j,v) \in E / 1 \leq j \leq k \} |$

$d_2(v) = | \{(v_j,v) \in E / k+1 \leq j \leq i \} |$

Entonces, se verifica que:

$K_2(u,v) = K_1(u,v)$ $\forall u,v \notin A$

$K_2(u,v) = K_1(u,v) + d_2(u)$ $\forall u \notin A \quad \forall v \in A$

$K_2(u,v) = K_1(u,v) + d_1(v)$ $\forall u \in A \quad \forall v \notin A$

$K_2(u,v) = K_1(u,v) + d_1(v) + d_2(u)$ $\forall u,v \in A$

Prueba

Sea G_1 el grafo resultante de eliminar de G los vértices $v_{i+1}, ...,v_n$ de L. Para u,v vértices de R, $K_1(u,v)$ es el número de intersecciones de aristas que llegan a u con aristas que llegan a v en G_1, considerando que u precede a v.

Sea G_2 el grafo resultante de añadir el vértice b entre las posiciones k y k+1 de L en G_1 junto con las aristas de G adyacentes a b.

Sea $u \in R / u \notin A$.

Si $v \notin A$ es evidente que $K_2(u,v) = K_1(u,v)$ ya que ninguna de las aristas incidentes con b que se han añadido a G_1 para obtener G_2 , son incidentes, ni con u ni con v.

Si $v \in A$, considerando el grafo G_2, si u precede a v, todas las aristas incidentes con u que parten de vértices por debajo de b, cortan a la arista (b, v). Dicha cantidad de aristas viene dada por el parámetro $d_2(u)$. Las aristas incidentes con u que parten de vértices por encima de b, no pueden cortar a la arista (b,v), tal y como muestra la figura 32. Por ello, $K_2(u,v) = K_1(u,v) + d_2(u)$

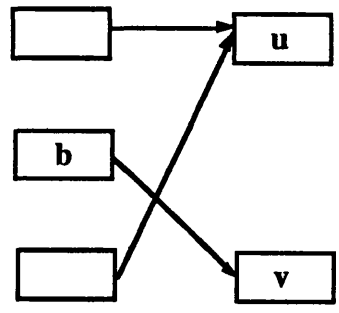


Figura 32

Sea $u \in R / u \in A$.

Si $v \notin A$, de la misma manera que en el caso anterior se comprueba que todas las aristas incidentes con v que parten de vértices por encima de b, cortan a la arista (b,u), a diferencia de las aristas incidentes con v que parten de vértices por debajo de b, por lo que, $K_2(u,v) = K_1(u,v) + d_1(v)$.

Si $v \in A$, considerando el grafo G_2, si u precede a v, todas las aristas incidentes con u que parten de vértices por debajo de b, cortan a la arista (b, v) mientras que las aristas incidentes con u que parten de vértices por encima de b, no pueden cortar a la arista (b,v), tal y como muestra la figura 33. Por otro lado, todas las aristas incidentes con v que parten de vértices por encima de b, cortan a la arista (b, u), mientras que las aristas incidentes con v que parten de vértices por debajo de b no pueden cortar a la arista (b,u). Por ello, $K_2(u,v) = K_1(u,v) + d_2(u) + d_1(v)$. ■

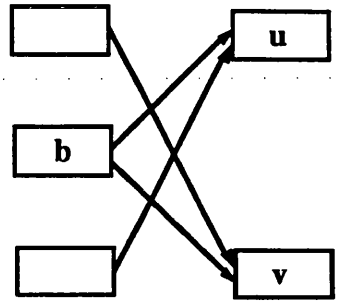


Figura 33

Este resultado también es válido cuando i=n, es decir cuando Q_1 es una clase de orden n y Q_2 es un nodo del primer nivel de ramificación.

Ejemplo:

Consideremos el siguiente ejemplo en el que la clase Q_1 está representada por la permutación (1,2) y eliminando el resto de vértices de L se obtiene el grafo G_1 de la figura 34. Sea Q_2 la clase representada por la permutación (1,2,3) y el grafo G_2 de la figura 35.

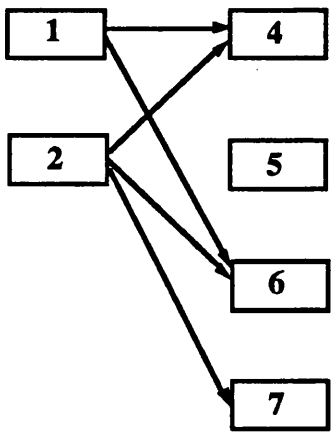


Figura 34. G_1

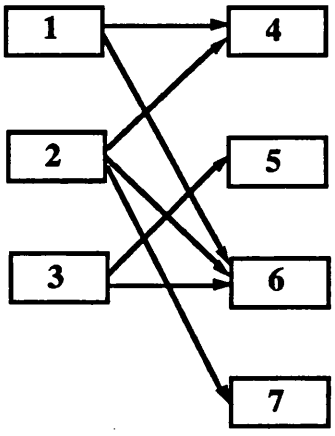


Figura 35. G_2

Al introducir el vértice 3 se tiene que A={5,6}. Aplicando del teorema se obtiene, por ejemplo:

$$K_2(5,6) = K_1(5,6) + d(6) = 0 + 2 = 2.$$
$$K_2(7,6) = K_1(7,6) = 1.$$

de este modo, se obtienen los pesos de las aristas de H_2 a partir de las de H_1.

2.5.3 UN TEST DE SATURACION DE NODOS.

Consideremos una solución óptima del problema y sean dos vértices u,v del mismo lado, consecutivos en la ordenación de dicha solución. Veamos qué propiedad cumplen dichos vértices.

TEOREMA 9

Sean u y v dos vértices consecutivos en una ordenación óptima. Si u precede a v, entonces: $k(u,v) \leq k(v,u)$.

Prueba

Supongamos que no fuera así. Entonces, bastaría con permutar las posiciones de u y v para obtener una solución con menor número de intersecciones. Dado que, al ser los vértices consecutivos, el permutar sus posiciones únicamente afecta al valor de k(u,v), manteniendose constantes los restantes k(i,j) con i,j vértices del mismo lado que u y v. ■

A partir de esta propiedad, se puede enunciar el siguiente test de saturación para los nodos en los que se han fijado algunos vértices del lado derecho.

> Sea P_i un nodo del nivel de ramificación $k > 2$ en el árbol de soluciones.
> Sea $(v_1, v_2,..,v_{k-2}, v_{k-1})$ el orden de los vértices de R fijados en el nodo.
>
> Si $k(v_{k-2}, v_{k-1}) > k(v_{k-1}, v_{k-2})$ entonces, saturar el nodo.

2.5.4 PREPROCESO

En este apartado vamos a estudiar técnicas para simplificar una jerarquía de dos niveles, permitiendo reducir el conjunto de nodos del árbol y, por lo tanto, haciendo la búsqueda más rápida.

VERTICES AISLADOS

En el caso trivial en el que existan vértices aislados, que tienen asociados una fila o columna de elementos nulos en la matriz de interconexión, se pueden eliminar a la hora de explorar las ordenaciones posibles. Dichos vértices se restituirán en cualquier posición una vez encontrada la solución óptima.

COMPONENTES CONEXAS

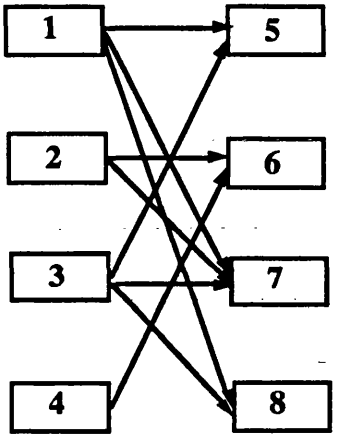
Si el grafo bipartido es disconexo, se estudiarán las componentes conexas por separado, ya que esto reduce en gran medida el número de ordenaciones, agrupando al final las soluciones óptimas.

FILAS Y/O COLUMNAS REPETIDAS

En [54] Warfield prueba el siguiente resultado: Sea N* la matriz de interconexión óptima de un grafo bipartido respecto al número de intersecciones. Sí N* tiene dos filas (columnas) iguales, entonces existe una matriz de interconexión con número mínimo de intersecciones donde dichas filas (o columnas) son mutuamente adyacentes.

Así pues, al buscar las ordenaciones óptimas de G, se pueden eliminar de la matriz de interconexión original las filas (o columnas) repetidas. Cuando se encuentre la solución óptima del problema reducido (en el que no están los vértices que representan a las filas o columnas duplicadas) se restituyen dichas filas (o columnas) en posiciones adyacentes a las de sus iguales.

Ejemplo: Consideremos el grafo siguiente con su matriz de interconexión:



$$\begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

Figura 36

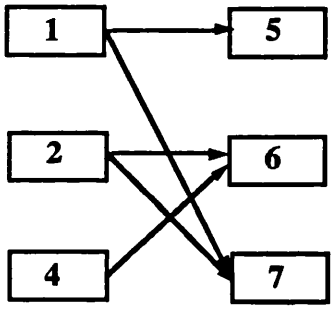


Figura 37.

Se elimina la fila 3 por ser igual a la 1 y la columna 4 por ser igual a la 1, obteniendo el grafo de la figura 30 con la siguiente matriz:

$$\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

A partir de este grafo se obtiene la solución óptima del problema reducido (figura 38), por lo que, para obtener la solución óptima del problema inicial, se sitúa la fila del vértice 3 junto a la del 1 y, la columna del vértice 8 junto a la del 5 (figura 39).

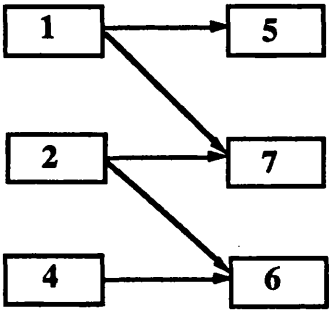


Figura 38.

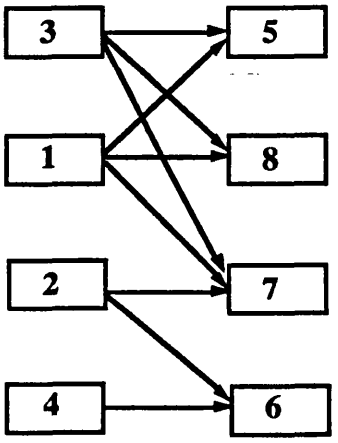


Figura 39

2.6 RESULTADOS

Para medir la bondad del algoritmo presentado en este capítulo se han efectuado pruebas sobre una colección de ejemplos aleatoriamente generados.

Los ejemplos se han generado considerando tres parámetros:
l = Número de vértices del lado izquierdo.
r = Número de vértices del lado derecho.
d = Densidad del grafo: Cociente entre el número de aristas del grafo y el número de aristas del grafo completo sobre el mismo número de vértices.

El algoritmo para generar grafos bipartidos aleatoriamente, construye un grafo en dos pasos a partir de los tres parámetros mencionados. En el primer paso garantiza el que todos los vértices tengan al menos grado uno. Para ello, para cada uno de los vértices del lado izquierdo, selecciona aleatoriamente un vértice del lado derecho y genera la arista correspondiente; después, para cada vértice del lado derecho al que no le llega ninguna arista, selecciona aleatoriamente un vértice del lado izquierdo y genera la arista correspondiente. En el segundo paso genera aleatoriamente el resto de aristas necesarias para que el grafo tenga la densidad fijada. Para ello, en cada iteración, selecciona aleatoriamente un vértice del lado izquierdo y uno del derecho; si no existe la arista entre ambos, la genera.

Todos los algoritmos que se mencionan en este apartado han sido programados en C y ejecutados en un ordenador personal compatible 486-66Mhz.

Como solución inicial del algoritmo se puede tomar cualquier ordenación del lado izquierdo y cualquier ordenación del lado derecho; por ejemplo, las proporcionadas aleatoriamente por el generador. Sin embargo, podemos mejorar fácilmente la elección de la solución inicial si tenemos en cuenta las siguientes ideas.

Consideremos un nodo de una clase del árbol de soluciones ampliado. La cota inferior del nodo depende, en parte, del número de aristas incidentes con los vértices fijados en el nodo. Por ello, considerando el modo en el que el algoritmo de ramificación y acotación realiza la exploración de los nodos del árbol, para que las cotas inferiores de los primeros nodos explorados sean lo mayores posible, se reordenarán en cada lado los vértices de la solución inicial por orden creciente de su grado.

Notar que el argumento empleado para los vértices del lado izquierdo es igualmente válido para los del lado derecho en los nodos de los primeros niveles de ramificación.

Para comprobar la validez de estas ideas, hemos realizado unas pruebas previas sobre dos grupos de grafos. En el primer grupo (G1) se han considerado quince ejemplos de nueve vértices a cada lado; en el segundo (G2), quince ejemplos de diez vértices a cada lado. En ambos grupos los grafos son de densidad igual a 0.3. Para cada uno de los ejemplos se han efectuado dos pruebas: una, aplicando el algoritmo con la solución inicial aleatoria (Prueba A) y otra, tomando como solución inicial la reordenación propuesta de dicha solución aleatoria (Prueba B).

Los resultados de estas pruebas aparecen reflejados en la siguiente tabla donde se consideran los siguientes parámetros:

A = Número total de nodos del árbol de soluciones ampliado.
E = Número de nodos examinados por el algoritmo.
T = Tiempo de ejecución del algoritmo (salvo que se indique lo contrario, son segundos).

Los valores de los parámetros que aparecen en la tabla son la media de los ejemplos probados.

		Prueba A		Prueba B	
	A	E	T	E	T
G1	$2{,}2 * 10^{11}$	31.705	15,5	3.961	1,6
G2	$2{,}2 * 10^{13}$	528.417	5 min.	15.057	7,2

Así pues, en adelante, se tomará como solución inicial del algoritmo de ramificación y acotación el grafo resultante de reordenar los vértices de ambos lados por orden decreciente de su grado, en el grafo generado aleatoriamente.

A continuación estudiemos el comportamiento del algoritmo en grafos en los que l+r es constante pero l y r toman diferentes valores.

La siguiente tabla muestra los resultados sobre cuatro grupos de grafos de 18 vértices. En cada grupo se han probado 15 ejemplos.

l	r	A	E	T
4	14	$3,5 * 10^{12}$	6,5	0,01
14	4	$3,5 * 10^{12}$	21.538	5,3
7	11	$3,4 * 10^{11}$	515	0,28
11	7	$3,4 * 10^{11}$	17.608	5,9

Considerando el modo en el que el algoritmo explora el árbol de soluciones, se explican los resultados de la tabla anterior, en la medida en que el número de clases del árbol depende de l. Así, teniendo en cuenta que las cotas inferiores de las clases son suficientemente buenas como para saturar a muchas de ellas, el número de nodos examinados y, por lo tanto, el tiempo de computación, será menor cuanto menor sea l. Además, hemos comprobado que en la mayoría de los ejemplos anteriores de 18 vértices, el algoritmo comienza a saturar clases en el nivel 4.

En adelante, dado un grafo en donde $l \neq r$, supondremos que $l<r$, ya que de no ser así, dada la simetría del problema, bastaría con intercambiar los papeles de ambos conjuntos.

Dado que la densidad del grafo es un factor determinante en el número de operaciones que realiza el algoritmo en cada nodo, y por lo tanto, en los tiempos de computación, hemos considerado oportuno dividir los grafos estudiados en tres clases: Primera clase de densidad 0.3, segunda clase de densidad 0.5 y tercera clase de densidad 0.7. Hemos de notar que, usualmente, los grafos que representan a proyectos son de densidad baja. Por ejemplo, un grafo con veinte vértices con una media de tres o cuatro aristas incidentes con cada vértice tiene una densidad aproximada de 0.3.

Los ejemplos estudiados se han dividido en grupos según el número de vértices del grafo. Dentro de cada grupo se han distinguido diferentes subgrupos según los valores de l y r. Para cada subgrupo se han considerado 10 ejemplos.

Además de los parámetros anteriormente mencionados, consideraremos los siguientes:

N = Número de vértices del grafo.

S = Número total de soluciones del problema (ordenaciones posibles: $l! * r!$).

T_{min} = Tiempo mínimo de ejecución del algoritmo en los 10 ejemplos del subgrupo.

T_{max} = Tiempo máximo de ejecución del algoritmo en los 10 ejemplos del subgrupo.

El rango de los tamaños de los ejemplos que muestran las siguientes tablas, ha sido determinado en función de los tiempos de computación. Así en densidad 0.3, no se han probado ejemplos por debajo de 18 vértices ya que el tiempo de ejecución era de 0 segundos[2]; y no se han probado ejemplos con más de 26 vértices ya que el tiempo de ejecución de algunas instancias del problema con 26 vértices era de 6 horas.

Grafos con densidad 0.3

N	l	r	S	A	E	T	T_{min}	T_{max}
18	5	13	$7,4 * 10^{11}$	$1,2 * 10^{12}$	44	0,04	0	0,06
	7	11	$2 * 10^{11}$	$3,4 * 10^{11}$	515	0,28	0,11	0,39
	9	9	$1,3 * 10^{11}$	$2,2 * 10^{11}$	3.961	1,6	0,38	1,76
20	6	14	$6,2 * 10^{13}$	$1 * 10^{14}$	124	0,09	0,05	0,11
	8	12	$1,9 * 10^{13}$	$3,3 * 10^{13}$	1.788	1,07	0,6	2,2
	10	10	$1,3 * 10^{13}$	$2,2 * 10^{13}$	15.057	7,2	5,9	8,9
22	6	16	$1,5 * 10^{16}$	$2,5 * 10^{16}$	201	0,19	0,06	0,28
	9	13	$2,2 * 10^{15}$	$3,8 * 10^{15}$	15.694	10,9	6,3	15,4
	11	11	$1,5 * 10^{15}$	$2,7 * 10^{15}$	89.020	49,8	19,2	1m 13s
24	8	16	$8,4 * 10^{17}$	$7,4 * 10^{15}$	13.863	5,7	1,4	14,6
	10	14	$3,1 * 10^{17}$	$5,4 * 10^{17}$	114.284	1m 31s	13	3m 10s
	12	12	$2,2 * 10^{17}$	$3,9 * 10^{17}$	203.396	2m. 17s	1m 15s	3m 18s
26	9	17	$1,2 * 10^{20}$	$2,2 * 10^{20}$	25.070	20	9	40
	11	15	$5,2 * 10^{19}$	$8,9 * 10^{19}$	234.368	3m 18s	48	5m 30s
	13	13	$3,8 * 10^{19}$	$6,6 * 10^{19}$	$9 * 10^{6}$	1h 32s	46m 32s	6h 28m

[2] La precisión es de décimas de segundos.

Grafos con densidad 0.5

N	l	r	S	A	E	T	T_{min}	T_{max}
18	5	13	$7,4 * 10^{11}$	$1,2 * 10^{12}$	146	0,08	0,05	0,28
	7	11	$2 * 10^{11}$	$3,4 * 10^{11}$	1.896	1,6	0,28	2,09
	9	9	$1,3 * 10^{11}$	$2,2 * 10^{11}$	16.661	9	2,47	12,1
20	6	14	$6,2 * 10^{13}$	$1 * 10^{14}$	300	0,3	0,1	0,55
	8	12	$1,9 * 10^{13}$	$3,3 * 10^{13}$	7.500	6,1	4	10,5
	10	10	$1,3 * 10^{13}$	$2,2 * 10^{13}$	115.695	1m 14s	27,5	1m 35s
22	6	16	$1,5 * 10^{16}$	$2,5 * 10^{16}$	3.408	2,1	0,3	7,3
	9	13	$2,2 * 10^{15}$	$3,8 * 10^{15}$	68.200	1m 5s	14	1m 40s
	11	11	$1,5 * 10^{15}$	$2,7 * 10^{15}$	$1,9*10^{6}$	30m 14s	17m	46m
24	8	16	$8,4 * 10^{17}$	$7,4 * 10^{15}$	13.745	19,4	11,8	28
	10	14	$3,1 * 10^{17}$	$5,4 * 10^{17}$	331.443	5m 30s	3m 39s	11m
	12	12	$2,2 * 10^{17}$	$3,9 * 10^{17}$	$15 * 10^{6}$	4 h	3 h	4h 50m

Grafos con densidad 0.7

N	l	r	S	A	E	T	T_{min}	T_{max}
18	5	13	$7,4 * 10^{11}$	$1,2 * 10^{12}$	93	0,10	0,05	0,11
	7	11	$2 * 10^{11}$	$3,4 * 10^{11}$	2713	2	1,53	2,64
	9	9	$1,3 * 10^{11}$	$2,2 * 10^{11}$	103.212	1m 10s	30,9	2 m
20	6	14	$6,2 * 10^{13}$	$1 * 10^{14}$	408	0,5	0,38	0,55
	8	12	$1,9 * 10^{13}$	$3,3 * 10^{13}$	19.027	20	8,45	24,8
	10	10	$1,3 * 10^{13}$	$2,2 * 10^{13}$	$1,4*10^{6}$	20 m	14 m	23m 32s
22	6	16	$1,5 * 10^{16}$	$2,5 * 10^{16}$	657	0,9	0,7	1,2
	9	13	$2,2 * 10^{15}$	$3,8 * 10^{15}$	117.280	2m 18s	47 s	3m 50s
	11	11	$1,5 * 10^{15}$	$2,7 * 10^{15}$	$5,9 * 10^{6}$	1h 30m	1h 9m	1h 50m
24	8	16	$8,4 * 10^{17}$	$7,4 * 10^{15}$	23.005	39,1	37	41
	10	14	$3,1 * 10^{17}$	$5,4 * 10^{17}$	$1,9 * 10^{6}$	45 m	43 m	47 m
	12	12	$2,2 * 10^{17}$	$3,9 * 10^{17}$	$38 * 10^{6}$	11 h	10h 9m	12 h

Respecto al comportamiento del algoritmo es importante destacar las siguientes observaciones: La potencia de las cotas inferiores de las clases, junto con la reordenación de vértices en la solución inicial, hace que se tenga que explorar únicamente una pequeñísima parte de los nodos del primer nivel de ramificación. Además, en el 80% de los nodos explorados en el primer nivel de ramificación, el procedimiento de obtención de la mejor solución asociada al nodo (apartado 2.3) produjo directamente una solución, permitiendo, de este modo, saturar el nodo.

A la vista de la diferencia entre el número total de nodos del árbol de soluciones (A) (o bien el número de soluciones posibles (S)) y los nodos examinados (E), así como los tiempos de computación, podemos afirmar que el algoritmo exacto presentado en este capítulo tiene un funcionamiento muy satisfactorio. No obstante, dada la naturaleza combinatoria del problema, el conjunto de soluciones posibles para grafos de más de 25 vértices es de un tamaño tal que los tiempos de computación son excesivamente altos como para resolver de manera óptima el problema. Por ello, en el capítulo tercero se desarrolla un algoritmo heurístico para resolver el problema de minimizar el número de intersecciones de las aristas de un grafo acíclico. Además, en dicho capítulo consideraremos la solución del algoritmo heurístico como solución inicial para el "Branch & Bound", repitiendo, con dicha solución inicial, las pruebas que hemos realizado.

SEGUNDA SECCION: GRAFOS ACICLICOS

1 OBTENCION DE UNA JERARQUIA PROPIA

Dado un grafo acíclico cualquiera se puede transformar en una jerarquía mediante una simple asignación ordenada de sus vértices a columnas o capas, como muestra el siguiente algoritmo:

Algoritmo 2

Considerar el grafo G=(V,E) acíclico.
i=1.
Mientras V≠∅:
- Asignar todos los vértices con grado de entrada 0 a la columna i.
- Eliminar de V todos los vértices asignados.
- Eliminar de E todas las aristas incidentes con los vértices asignados.
- i = i+1.

TEOREMA 10

Sea G=(V,E,n) la jerarquía proporcionada por el algoritmo descrito a partir del grafo acíclico G=(V,E). Entonces, considerando el conjunto de las jerarquías que se pueden obtener a partir de G, n es mínimo.

Prueba

En primer lugar consideremos que toda jerarquía tiene la propiedad de que todas las aristas tiene un vértice inicial con número de columna estrictamente menor que el del vértice final. Para probar el resultado vamos a ver que la jerarquía obtenida tiene al menos una cadena de vértices, comenzando en el primer nivel y terminando en el n, con vértices en todos los niveles; con lo cual quedará probado el resultado ya que dicha cadena "no cabría" en una jerarquía con menor número de niveles.

Puesto que en el algoritmo un vértice se coloca en la jerarquía tan pronto como han sido colocados sus predecesores, todo vértice en el nivel i tiene al menos un predecesor en el nivel i-1, por que de no ser así dicho vértice estaría en un nivel anterior.

Consideremos un vértice v_n en la capa n. Sea v_{n-1} un predecesor en el nivel n-1. Dado v_{n-1}, sea v_{n-2} un predecesor en el nivel n-2. Por este procedimiento se llega hasta un vértice v_1 en el nivel 1, consiguiendo la cadena (v_1 v_2 v_3 .... v_n) donde cada v_i pertenece al nivel i.■

Warfield [53] considera el grafo ya estructurado como jerarquía propia de n niveles, aunque señala que para solucionar el problema de las aristas de longitud mayor que 1 (long span edges) basta con añadir vértices y aristas ficticios. Una vez obtenida la jerarquía, para hacerla propia, se sustituyen las aristas de longitud mayor que uno por cadenas de vértices y aristas con un número igual a la longitud de la arista original menos uno, como muestra el siguiente algoritmo.

Sea G=(V,E,h) una jerarquía de h niveles donde $V= V_1 \cup V_2 \cup....\cup V_n$ según la definición. Llamaremos Temp_E al conjunto temporal de aristas del grafo G, que inicialmente es igual a E, de donde se irán tomando, examinando y eliminando dichas aristas. Análogamente haremos los mismo con Temp_V. En VFIC se guardan los vértices ficticios, y en AFIC las aristas ficticias. Notar que cuando se cambia una arista de longitud mayor que uno por varias de longitud igual a uno, no sólo se añaden las ficticias sino que además se elimina la original, por ello, si denominamos a la jerarquía propia resultante de dichas transformaciones como G*=(V*,E*,h) se tiene que:

$V^* = V \cup VFIC.$

$E^* = AFIC \cup E - \{$ Aristas originales de longitud mayor que 1$\}$

Por último señalar que si $| V | = n$ los vértices ficticios serán numerados como {n+1,n+2,...}, por lo que en adelante utilizaremos la siguiente notación para referirnos a una jerarquía propia de h niveles con vértices ficticios: G*=(V*,n,E*,h), donde h es el número de niveles y n es el número de vértices originales, con lo cual $| V^* | - n$ es el número de vértices ficticios.

Algoritmo 3

```
INICIALIZACION:

VFIC = AFIC =ø
Temp_E = E.
Temp_V = V.
cont= | V |

WHILE ( Temp_E = ø )
{
        Tomar (a,b) ∈ Temp_E.
        Temp_E = Temp_E - {(a,b)}
        Sean i , j  /  a ∈ Vi y b ∈ Vj
        tramos = j-i.
        if(tramos > 1)                          /* Sustituir arista */
        {
                        E = E - {(a,b)}         /* Primer tramo */
```

```
                    VFIC = VFIC ∪ {cont+1}
                    AFIC = AFIC ∪ {(a,cont+1)}
                    cont = cont + 1.
                    while(tramos > 2)              /* Hay más de dos tramos*/
                    {
                        tramos =tramos -1.
                        VFIC = VFIC ∪ {cont+1}
                        AFIC = AFIC ∪ {(cont,cont+1)}
                        cont = cont + 1.
                    }
                    AFIC = AFIC ∪ {(cont,b)}       /* Último tramo*/
            }
    }
```

Veamos en un ejemplo como actúan los dos algoritmos descritos sobre un grafo acíclico cualquiera, convirtiéndolo primero en una jerarquía y luego haciéndola propia.

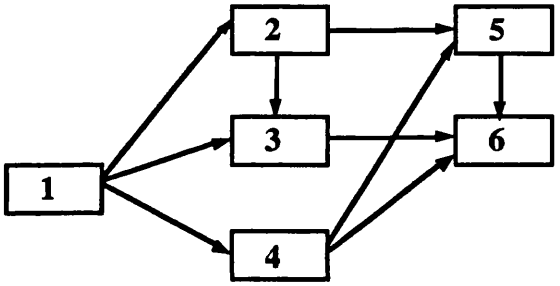


Figura 40. Grafo acíclico original

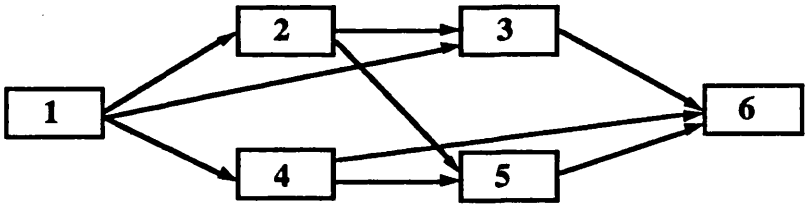


Figura 41. Jerarquía

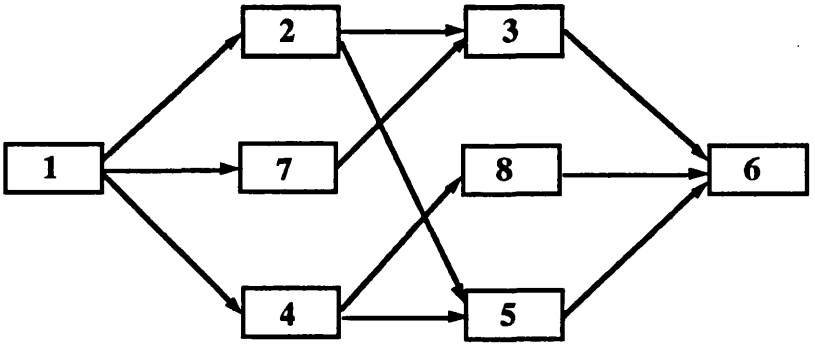


Figura 42. Jerarquía propia

La sustitución de las aristas de longitud mayor que uno por cadenas de vértices y aristas tiene los siguientes aspectos destacables:

a.-Mantiene el número de intersecciones del grafo original.

b.-Permite evaluar el número de intersecciones mediante el mismo procedimiento que en las jerarquías de dos niveles, ya que, todas las aristas van de un nivel al siguiente.

c.- Aumenta el tamaño del grafo tanto en vértices como en aristas.

2 CALCULO DEL NUMERO DE INTERSECCIONES

En este capítulo se denotará a las jerarquías por G=(V,E,n), donde V es el conjunto de vértices, E el de aristas y n el número de niveles. V_1, V_2, .., V_n representarán dichos niveles y, m_i será el cardinal del nivel V_i.

En el capítulo anterior se definió la matriz de interconexión en jerarquías de dos niveles. De manera natural se extiende esta definición a jerarquías de n niveles en las que, para cada par de niveles consecutivos se considera una matriz de interconexión.

Definición

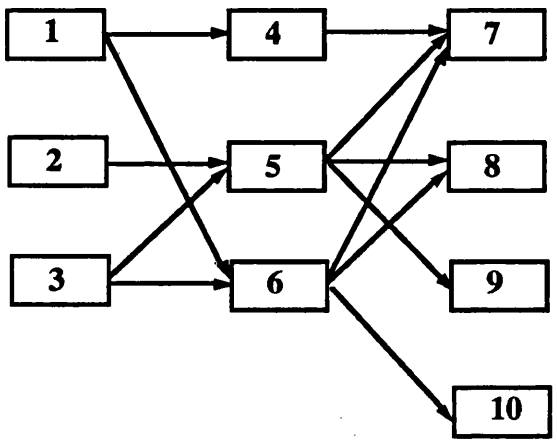
Sea G=(V,E,n) una jerarquía propia de n niveles. Para cada par de niveles consecutivos V_i V_{i+1} se define la matriz de interconexión N_i de dimensiones $m_i * m_{i+1}$ asociada a unas ordenaciones de dichos niveles como:

$N_i(j,k)=1$ Si existe un arco del j-ésino vértice de V_i al k-ésimo vértice de V_{i+1}.

$N_i(j,k)=0$ En otro caso.

De este modo, una jerarquía propia de n niveles con unas ordenaciones define n-1 matrices de interconexión.

Ejemplo:



$$N_1 = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}$$

$$N_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{bmatrix}$$

Figura 43

Dada una jerarquía G=(V,E,n) se define, de manera análoga al capítulo anterior, K(i,j) para i,j en el mismo nivel V_r, como el número de intersecciones de aristas que parten del i-ésimo vértice de V_r con aristas que parten del j-ésimo vértice de V_r, en donde

i<j. Dicho valor se puede evaluar a partir de la matriz de interconexión $N_r=(n_{ij}^r)$ de los niveles r,r+1.

$$K(i,j) = \sum_{k=1}^{m_{i+1}-1} \sum_{k=1}^{m_{i+1}} n_{jk}^r \, n_{is}^r$$

A partir de esta expresión, para evaluar el número total de intersecciones de las aristas entre dos niveles consecutivos r, r+1, basta con considerar:

$$I_r = \sum_{i=1}^{m_i-1} \sum_{j=i+1}^{m_i} K(i,j)$$

Por lo que el número total de intersecciones de aristas de la jerarquía de n niveles viene dado por la expresión:

$$I = \sum_{r=1}^{n-1} I_r$$

3 ARBOL DE SOLUCIONES.

Dada una jerarqía propia de n niveles G=(V,E,n), toda solución del problema consta de n ordenaciones. La forma en la que dichas ordenaciones son generadas y particionadas en el árbol de soluciones es la siguiente:

Cada nivel de ramificación del árbol de soluciones viene dado por las posibles ordenaciones de un nivel de la jerarquía propia. A partir del nodo inicial se ramifica en tantos nodos como ordenaciones posibles del nivel V_1 de la jerarquía (sin considerar las ordenaciones opuestas). Cada uno de dichos nodos se ramifica en tantos nodos como ordenaciones posibles del nivel V_2. Este proceso se repite hasta llegar al nivel V_{n-1} en donde los nodos se ramifican de modo similar al diseñado en el árbol de soluciones del capítulo anterior.

Cada nodo del nivel de ramificación n-1 se ramifica en m_n nodos, en donde se fija el vértice de V_n que será el primero de la ordenación; los cuales a su vez se ramifican en m_n -1 nodos, en los que se fija el segundo vértice de V_n. Este procedimiento se sigue hasta asignar todos los vértices de V_n.

Pasamos a mostrar, el árbol de soluciones de una jerarquía propia de cuatro niveles, en donde V_1={1,2,3}, V_2={4,5,6,7}, V_3={8,9} y V_4={10,11,12}.

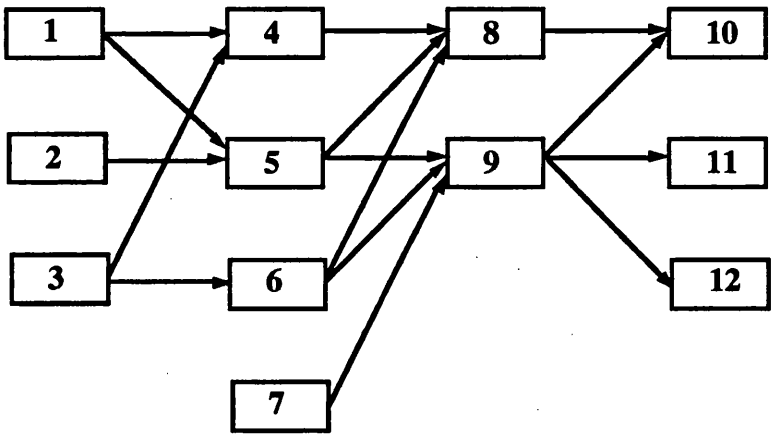
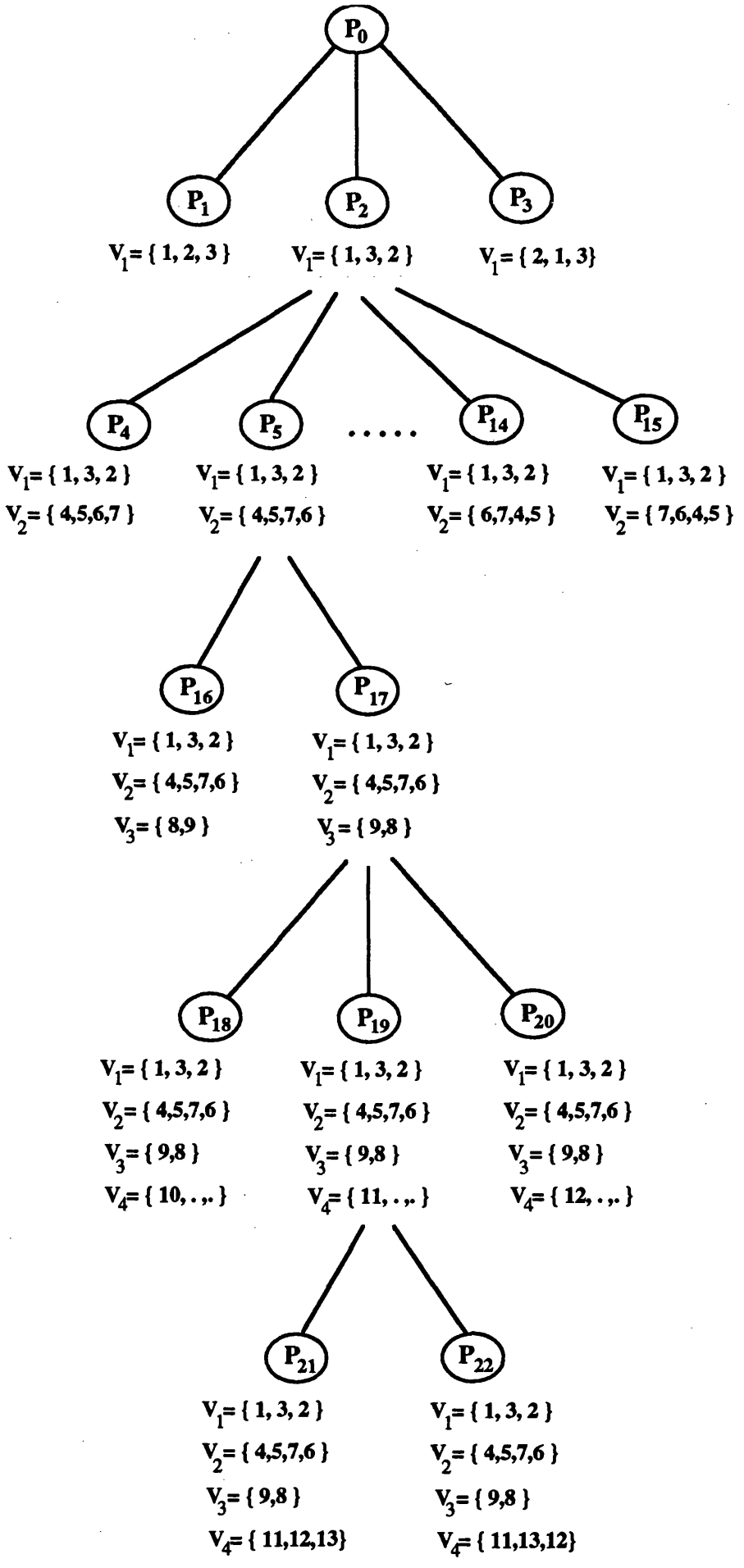


Figura 44


P0
P1
P2
P3
V1= { 1, 2, 3 }
V1= { 1, 3, 2 }
V1= { 2, 1, 3}
P4
P5
.....
P14
P15
V1= { 1, 3, 2 }
V2= { 4,5,6,7 }
V1= { 1, 3, 2 }
V2= { 4,5,7,6 }
V1= { 1, 3, 2 }
V2= { 6,7,4,5 }
V1= { 1, 3, 2 }
V2= { 7,6,4,5 }
P16
P17
V1= { 1, 3, 2 }
V2= { 4,5,7,6 }
V3= { 8,9 }
V1= { 1, 3, 2 }
V2= { 4,5,7,6 }
V3= { 9,8 }
P18
P19
P20
V1= { 1, 3, 2 }
V2= { 4,5,7,6 }
V3= { 9,8 }
V4= { 10, . ,. }
V1= { 1, 3, 2 }
V2= { 4,5,7,6 }
V3= { 9,8 }
V4= { 11, . ,. }
V1= { 1, 3, 2 }
V2= { 4,5,7,6 }
V3= { 9,8 }
V4= { 12, . ,. }
P21
P22
V1= { 1, 3, 2 }
V2= { 4,5,7,6 }
V3= { 9,8 }
V4= { 11,12,13}
V1= { 1, 3, 2 }
V2= { 4,5,7,6 }
V3= { 9,8 }
V4= { 11,13,12}

4 COTAS INFERIORES DE UN NODO.

4.1 NODOS DE UN NIVEL DE RAMIFICACION MENOR QUE N

Considerando que un nodo del segundo nivel de ramificación (por ejemplo, el P_4) representa a todas las soluciones del problema, en las que las ordenaciones del primer y segundo nivel de la jararquía son las fijadas en el nodo, se puede evaluar de forma exacta, para esas ordenaciones concretas, el número de intersecciones generado por las aristas entre el primer y segundo nivel. Dicho número es una cota inferior al número total de intersecciones de cualquier solución en ese nodo.

Dado un nodo P en el nivel de ramificación j, con j menor que n, tomando a I_i como el número de intersecciones de las aristas entre el nivel i y el i+1 de la jerarquía, la siguiente expresión proporciona la cota inferior del nodo.

$$K = \sum_{r=1}^{i-1} I_r$$

De este manera, cuanto más profundo es el nivel del árbol de soluciones mayor es el número de niveles de la jerarquía en los que la ordenación está fijada y, por lo tanto, mayor es la cota inferior.

4.2 NODOS DE UN NIVEL DE RAMIFICACION MAYOR O IGUAL QUE N

En cada nodo a partir del nivel de ramificación n, están fijadas las ordenaciones de los n-1 primer niveles de la jerarquía y algunos vértices del nivel n. Así pues, se pueden considerar dos cotas inferiores, de manera análoga a como se hizo en el capítulo anterior, mediante la suma de cuatro parámetros ($k = k_1 + k_2 + k_3 + k_4$).

K_1 es el número de intersecciones producidas por las aristas incidentes con los vértices ya fijados desde el nivel 1 al n-1 de la jerarquía.

$$K_1 = \sum_{r=1}^{n-2} I_r$$

Considerando las aristas entre el nivel n-1 y el n de la jerarquía, se obtienen los tres parámetros siguientes:

K_2 es el número de intersecciones producido por las aristas incidentes con los vértices ya fijados del nivel n.

K_3 es el número de intersecciones producido por las parejas de aristas, en las que una es incidente con un vértice fijado y la otra con uno no fijado (ambos en el nivel n).

K_4 es una cota inferior al número de intersecciones de aristas incidentes con los vértices no fijados en el nodo.

En la primera cota inferior K_4 es el peso del camino hamiltoniano de mínimo peso del grafo auxiliar H dirigido y completo, formado por los vértices no fijados. En dicho grafo los pesos de las aristas se calculan como se mostró anteriormente.

En la segunda cota inferior K_4 se obtiene tomando para cada par de aristas que unen dos vértices la de coste menor, y después sumando dichos costes para todos los pares de vértices del grafo auxiliar H.

Ejemplo.

Sea el nodo P_{19} del árbol de soluciones del ejemplo anterior, en el que las ordenaciones son V_1={1,3,2}, V_2={4,5,7,6}, V_3={9,8} y V_4={11,..,.}, se tiene que:

Tomando en consideración las aristas entre V_1, V_2 y V_2, V_3 se obtiene K_1=7. Puesto que sólo se ha fijado el vértice once del cuarto nivel, K_2=0. Al ser el 11 el primer vértice del nivel, los no fijados irán detrás de él, por lo que K_3=k(11,10)+k(11,12)=0. Dado que sólo quedan los vértices diez y doce por asignar, el grafo H únicamente tendrá estos dos vértices con las dos aristas dirigidas entre ambos. Los costes de dichas aristas (k(10,12) y k(12,10)), coinciden y valen uno, así pues, tanto en la primera cota inferior como en la segunda k_4=1. Luego en este nodo la cota inferior es K=8.

CAPITULO III

Algoritmo Heurístico "Tabu Thresholding"

En este capítulo se estudia el mismo problema que en el anterior, desarrollando un algoritmo heurístico para resolverlo. El problema es abordado en dos fases: En la primera se trabaja únicamente con jerarquías propias de dos niveles, para después generalizar los resultados obtenidos a jerarquías propias de cualquier número de niveles. En el segundo capítulo se mostró cómo todo grafo acíclico se puede transformar en una jerarquía propia, con lo cual el algoritmo es aplicable a grafos acíclicos.

Utilizando las técnicas Tabu Thresholding [21], desarrollamos un algoritmo heurístico para resolver el problema citado. La eficacia del algoritmo se ha medido comparando los resultados con los que proporcionan los más eficientes aparecidos en la literatura, sobre una colección de grafos generados aleatoriamente; constatando así la superioridad del algoritmo presentado. En todos los casos en los que se disponía de la solución óptima, se ha comprobado que el algoritmo presentado proporciona dicho óptimo.

Tomando como cota superior inicial en el algoritmo exacto la solución obtenida mediante el heurístico, se contrastan los resultados con los obtenidos en el capítulo segundo.

1 GRAFOS BIPARTIDOS.

1.1 DESCRIPCION DEL ALGORITMO.

El esquema que se ha utilizado para construir el algoritmo heurístico es el denominado Tabu Thresholding [21]. Como se comentó en el capítulo de introducción el esquema se basa en la definición de un conjunto de movimientos, que, asociados a cada solución del problema, permiten pasar de una solución a otra. El esquema consta de dos fases denominadas "Improving" y "Mixed", que se repiten de manera alternada hasta satisfacer un criterio de parada. A continuación, veamos como se definen cada uno de dichos conjuntos y fases en nuestro problema.

Definición del conjunto de movimientos.

Dada una jerarquía de dos niveles G=(L,R,E) y siendo l,r ordenaciones de los conjuntos L y R, se considera el conjunto ordenado M=L∪R en donde primero están los vértices de L según el orden l y a continuación los vértices de R según el orden r.

Dado un vértice $v \in M$ denotaremos por Move(v) al conjunto de movimientos asociados a v, el cual se define de la siguiente manera:

a) Si el vértice v es el primero de la ordenación (tanto si es de L como de R) el único movimiento que se considera es permutar v con el vértice siguiente en la ordenación. Notaremos a este movimiento como m(v)+.
Si $v \in L$ y $l(v)=1$ ó $v \in R$ y $r(v)=1$ ------> Move(v) = { m(v)+ }

b) Si el vértice v es el último de la ordenación (tanto si es de L como de R) el único movimiento que se considera es permutar v con el vértice anterior en la ordenación. Notaremos a este movimiento como m(v)-.
Si $v \in L$ y $l(v) = |L|$ ó $v \in R$ y $r(v)= |R|$ -----> Move(v) = { m(v)- }

c) Si el vértice v no es ni el primero ni el último en la ordenación consideraremos dos posibles movimientos: permutarlo con el vértice anterior en la ordenación o permutarlo con el vértice posterior. Notaremos al primer movimiento como m(v)- y al segundo como m(v)+.
Si $v \in L$ y $1<l(v)<|L|$ ó $v \in R$ y $1<r(v)<|R|$ -----> Move(v) = { m(v)-,m(v)+ }

Una vez seleccionado un vértice y el conjunto de movimientos asociado a dicho vértice, para evaluar la conveniencia o no de realizar un movimiento, basta con calcular la variación del número de intersecciones al aplicar el movimiento.

Sea $u \in L$ un vértice del apartado c anterior, y $v,z \in L$ los 2 vértices adyacentes a u / $l(v)<l(u)<l(z)$. La selección del movimiento que más disminuye el número de intersecciones se realiza calculando $K(v,u)-K(u,v)$ y $K(u,z)-K(z,u)$. Si ambas cantidades son negativas, esto indica que la ordenación actual es la mejor posible respecto a los vértices v,u y z. En caso contrario, se elige la mayor de ambas; lo cual indica el movimiento más conveniente. Así si $K(v,u)-K(u,v)>0$ y $K(v,u)-K(u,v) > K(u,z)-K(z,u)$ se intercambiarán las posiciones de u y v en la ordenación, obteniendo $l(u)<l(v)<l(z)$. Si el número de intersecciones del grafo era de I antes del cambio, después de éste será de $I-(K(v,u)-K(u,v)) < I$.

Si el vértice u pertenece a los apartados a) o b) la evaluación es más sencilla puesto que sólo hay un movimiento posible.

En el capítulo anterior se presentó un algoritmo para calcular K(u,v) y K(v,u), obteniendose un procedimiento evaluador del conjunto de moviminetos rápido y eficiente[1], mediante una única lectura de las listas de adyacencia de ambos vértices.

Descripción de la fase "Improving"

Se considera el conjunto M ordenado como la lista de candidatos que utiliza el algoritmo y es examinado mediante la estrategia *Block Random Order,* dividiendo el vector en intervalos.

Una vez seleccionado un intervalo se escoge un vértice de modo aleatorio, se identifica si pertenece a L o a R y se examinan los movimientos asociados a dicho nodo, ejecutando el que más disminuya el número de intersecciones. En el caso en que ninguno disminuya el número de intersecciones no se realiza movimiento.

[1] Los procedimientos de búsqueda de soluciones Tabu Search se basan en realizar gran cantidad de movimientos sobre el conjunto de soluciones posibles. Por ello, para que sean eficientes, la evaluación, selección y aplicación de dichos movimientos, han de requerir un esfuerzo computacional bajo.

En esta fase el algoritmo sólo permite realizar un movimiento cuando éste disminuya el número de intersecciones, el criterio de parada se basa en fijar un número máximo de iteraciones en las que no se produzca movimiento; puesto que se considerará que tras este número de iteraciones, en las que se ha explorado el conjunto M y no se ha encontrado ningún cambio que produzca una ordenación con menos intersecciones, la ordenación actual está suficientemente cerca del óptimo local como para detener esta fase. Así pues, la elección de dicho número máximo de iteraciones se hará para proporcionar esa cercanía al óptimo local.

A continuación veamos el esquema algorítmico de esta fase.

Algoritmo. Fase Improving

1.- Considerar los conjuntos L y R con un orden inicial dado (l,r) y construir M como una lista circular ordenada.
no_cambio=0.

2.- Sea I el intervalo formado por los α elementos siguientes (En la primera iteración tomar los α primeros).

3.- Seleccionar v aleatoriamente de I.

4.- Evaluar para cada elemento de Move(v) el número de cruces resultantes al aplicar dicho movimiento al grafo.

IF(algún movimiento disminuye el número de cruces)
Tomar el que más disminuya el número de cruces.
Actualizar la ordenación de l o r según corresponda.
Actualizar la ordenación de M.
no_cambio=0.
ELSE
No realizar movimiento.
no_cambio = no_cambio + 1.

5.- IF (no_cambio = tope1)
parar.
ELSE
ir a 2.

Descripción de la fase "Mixed"

Esta fase se ejecuta siempre después de la Improving, considerando por ello el vector M con el orden obtenido en la última iteración de la fase Improving. M es explorado con la estrategia *Full-Random-Order*: Se selecciona aleatoriamente un vértice de M y se evalúan los movimientos asociados a dicho vértice, tomando el mejor de todos respecto al valor de la función objetivo, incluso cuando este movimiento aumente el número de cruces.

El criterio de parada en esta fase viene dado por un número máximo de iteraciones. A continuación, veamos el esquema algorítmico de esta fase.

Algoritmo. Fase Mixed

1.- Considerar M con el orden resultante de la fase Improving.
n_it = 1.

2.- Selecciona v aleatoriamente de M.
n_it = n_it + 1.

3.- Evaluar para cada elemento de Move(v) el número de cruces resultantes al aplicar dicho movimiento al grafo.
Tomar el que más disminuya el número de cruces.
Actualizar la ordenación de l o r según corresponda.
Actualizar la ordenación de M.

4.- IF (La solución obtenida tiene menos cruces que la almacenada como mejor solución)
Guardar la solución actual como mejor solución.

5.- IF (n_it = tope2)
parar.
ELSE
ir a 2.

Puesto que el procedimiento especifica iterar alternativamente con ambas fases es importante notar que cuando se comienza una fase se mantiene en M el orden producido por la fase anterior.

Como en la fase Improving el algoritmo va mejorando las soluciones basta con almacenar al final de dicha fase la última solución, puesto que será la mejor; sin embargo

en la fase Mixed la solución puede mejorar o empeorar en cada iteración, con lo cual es necesario contrastar la solución actual con la mejor almacenada cada vez que se realice un movimiento, actualizando la solución almacenada como "mejor" cuando sea superada.

El criterio de parada global se ha establecido de modo similar al de la fase improving: Una vez terminada la fase improving se contrasta el número de cruces de la ordenación obtenida con el que se obtuvo en la anterior ejecución de dicha fase; el algoritmo se detiene cuando tras un número de iteraciones completas igual a tope3 la solución no ha mejorado.

El siguiente esquema muestra el funcionamiento del algoritmo global:

Algoritmo Global

```
1.- mejor_cortes = número de cruces de la ordenación original.
    cambio_total = 0.

2.- While( cambio_total < tope3 )
    {
            2.1 Realizar la fase Improving.
                Sea cortes = número de cruces en la ordenación obtenida.
                IF ( cortes < mejor_cortes )
                        mejor_cortes = cortes.
                        cambio_total = 0.
                ELSE
                        cambio_total = cambio_total + 1.

            2.2 Realizar la fase Mixed.
                Sea cortes = número de cruces de la mejor ordenación de todas las
                             obtenidas en la fase Mixed anterior.
                IF ( cortes < mejor_cortes )
                        mejor_cortes = cortes.
}
```

1.2 OTROS ALGORITMOS PROBADOS.

Para llegar a formular el algoritmo tal y como se ha descrito hemos realizado numerosas pruebas con diferentes estrategias algorítmicas. A continuación, describiremos las más interesantes de las probadas aunque todas ellas proporcionan peores resultados que la anteriormente descrita (y que llamaremos Algoritmo 1).

ALGORITMO 2

Igual al algoritmo 1 salvo el punto 3 de la fase improving que ahora es: Tomar de I aquel v tal que al considerar el conjunto de movimientos proporcione el mejor cambio posible (i.e. disminuya más el número de intersecciones).

ALGORITMO 3

La fase improving igual que la del algoritmo 1 y en la mixed se examina todos los vértices de M de manera ordenada del primero al último, realizando siempre cambio aunque empeore la solución.

ALGORITMO 4

Igual que el algoritmo 3 pero aquí se examinan todos los vértices de forma aleatoria sin repetición.

ALGORITMO 5

Como el algoritmo 1 salvo que en el punto 3 de la improving phase ahora se tiene: Asignar a cada uno de los cinco elementos un peso según el número de intersecciones que disminuya su mejor movimiento y elegir con estos pesos de forma aleatoria un elemento.

1.3 RESULTADOS

Para medir la bondad del algoritmo propuesto se han efectuado pruebas sobre una colección de ejemplos generados aleatoriamente mediante el algoritmo descrito en el apartado de resultados del segundo capítulo.

Los resultados del algoritmo han sido comparados con la solución óptima, cuando se disponía de ella. Asimismo, se han comparado con los resultados que se obtienen con los algoritmos *Greedy Switching y Splitting,* presentados por Eades y Kelly [9] y que según muestran dichos autores porporcionan soluciones muy cercanas al óptimo. Según nuestras referencias, estos son los mejores algoritmos heurísticos aparecidos en la literatura para resolver el problema de minimizar intersecciones en jerarquías de dos capas.

Evaluaremos los resultados mediante los siguientes parámetros:

inicial = número inicial de intersecciones.
óptimo = número de intersecciones de la solución óptima.
T = tiempo de ejecución en segundos del algoritmo 1 para encontrar la solución.
tabu = número de intersecciones de la solución del algoritmo 1.
sw = número de intersecciones de la solución dada por el algoritmo Greedy Switching.
split = número de intersecciones de la solución dada por el algoritmo Splitting.

El algoritmo 1 ha sido programado en C y ejecutado en un ordenador personal compatible 486-66Mhz. Para cada una de las clases consideradas se han probado diez ejemplos, por lo que los parámetros que aparecen en la siguiente tabla son la media de los parámetros descritos sobre ellos. En todos los problemas se ha considerado que el número de vértices era el mismo en ambos lados.

Tras realizar numerosas pruebas, contrastando la solución con el óptimo, se han establecido los valores: $\alpha = 5$, tope1=25, tope2=6*(|M| + |E|) y tope3=50 en el algoritmo 1. Sin embargo, es importante destacar que con unos valores de tope1, tope2 y tope3 mucho menores, en el 90% de los casos se obtenía el óptimo.

Los ejemplos se han dividido en tres grupos según su densidad, al igual que se hizo en el capítulo segundo: Los de densidad baja e igual a 0.3, los de media igual a 0.5 y los de alta igual a 0.7. En todos los casos, la solución inicial es la proporcionada por el generador aleatorio.

Grafos con densidad 0.3

N	Inicial	Optimo	Tabu	T	Sw	Split
18	95	29,3	29,3	1,1	33,2	31,4
20	160	58	58	1,3	70	67,2
22	280,4	99,3	99,3	2,5	112,8	107
24	400	130	130	3	148	135
26	560	201,5	201,5	3,8	219,8	218
30	880	---	455,5	6,2	470	468,9
40	3150	---	1715	20	1890	1837,2
50	7830	---	4700	1m 5s	4790	4783,2
100	130757	---	97453	8m 14s	108490	101386

Grafos con densidad 0.5

N	Inicial	Optimo	Tabu	T	Sw	Split
18	297	148	148	3,1	157,9	156
20	480	275	275	6,5	290	287
22	730,5	429,3	429,3	6,9	435,2	435
24	1100,1	622	622	8,4	630	627,5
30	2839	---	1772	25	1830,3	1790
40	8840	---	6190	59,4	6200	6199,7
50	22100	---	16250	3m 17s	16500	16401
100	375332	---	306598	1h 5m	325000	321897

Grafos con densidad 0.7

N	Inicial	Optimo	Tabu	T	Sw	Split
18	297	148	148	3,1	157,9	156
20	480	275	275	6,5	290	287
22	730,5	429,3	429,3	6,9	435,2	435
24	1100,1	622	622	8,4	630	627,5
30	5200	---	4070	48	4400	4230
40	17890	---	14150	3m 10s	18000	17900
50	46095	---	36248	8m	36780	36400
100	740120	---	647830	2h 20m	690230	675000

Aunque los parámetros de la tabla son la media sobre cada clase, es importante señalar que **en todos los ejemplos** en los que se dispone del óptimo, **la solución del algoritmo heurístico es la óptima.** Asimismo, en todos los ejemplos probados, la solución del algoritmo tabu mejoró a la de los otros dos algoritmos heurísticos considerados.

Para comprobar si la distribución inicial influye o no en la solución dada por el algoritmo, hemos tomado cuatro de los ejemplos anteriores y para cada uno de ellos hemos considerado diez permutaciones distintas de los vértices en las dos capas. Tomando dichas permutaciones como distribuciones iniciales hemos hecho actuar el algoritmo sobre todas ellas y el resultado siempre ha tenido el mismo número de intersecciones, lo cual parece indicar que: La solución proporcionada por el algoritmo es independiente de la distribución inicial de los vértices, y por lo tanto del número original de intersecciones de las aristas.

1.4. ALGORITMO EXACTO CON COTA SUPERIOR INICIAL TABU

En este apartado vamos a comparar los resultados obtenidos mediante el algoritmo exacto descrito en el capítulo segundo[2] con los obtenidos cuando la cota superior inicial es el número de intersecciones de la solución proporcionada por el algoritmo tabu.

En ambos casos consideraremos que la solución inicial es la generada aleatoriamente y reordenada según los grados, tal y como se mencionó en el apartado de resultados del capítulo segundo. Evaluaremos los resultados mediante los siguientes parámetros:

N = número de vértices del grafo.
E1 = número de nodos examinados por el algoritmo exacto con cota superior inicial igual al número de intersecciones de la solución inicial aleatoria reordenada.
T1 = tiempo de ejecución en segundos del algoritmo anterior.
E2 = número de nodos examinados por el algoritmo exacto con cota superior inicial igual al número de intersecciones de la solución del algoritmo tabu.
T2 = tiempo de ejecución en segundos del algoritmo anterior.

Para realizar este estudio consideraremos los grafos con densidad 0.5 y únicamente aquellos en los que l=r, ya que son los que más tiempo de computación requieren. Para cada clase se han probado diez ejemplos.

Grafos con densidad 0.5

N	E1	T1	E2	T2
18	16.661	9	9875	5,8
20	115.695	1m 14s	104.841	58
22	$1,9 * 10^6$	30m 14s	$1,7 * 10^6$	22m
24	$15 * 10^6$	4 h.	$14,5 * 10^6$	3h 30m

Como se puede ver en la tabla, al considerar como cota superior inicial el número de cortes de la solución del algoritmo tabu, los tiempos de computación se reducen en torno a un 25%.

[2] Donde la cota superior inicial era el número de intersecciones de la solución aleatoria inicial.

2 JERARQUIAS DE N NIVELES.

Dado que el conjunto de movimientos posibles ha sido asociado a los vértices del grafo, existe la posibilidad de generalizar el algoritmo a jerarquías de n niveles manteniendo la definición de movimiento expuesta para jerarquías de dos niveles, así como el criterio de selección del movimiento en cada fase. Lo único que diferirá será la evaluación del número de intersecciones de las aristas involucradas en un movimiento.

2.1 EVALUACION DEL NUMERO DE INTERSECCIONES.

El número del nivel de un vértice y su orden dentro de dicho nivel, vienen definidos por los siguientes parámetros:

Definición

Dado un vértice v se denota por **x(v)** al número de columna (abcisa) de la plantilla que ocupa el vértice v, y por **y(v)** al número de fila (ordenada).

Ejemplo:

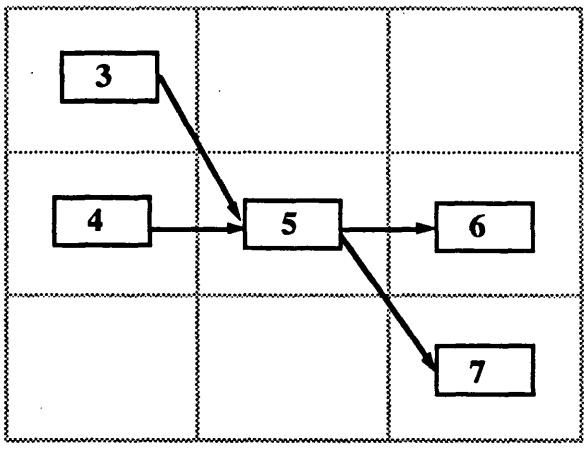


Figura 1. Ejemplo de plantilla de trabajo

Así en el ejemplo de la figura 1, se tiene: x(4)=1, y(3)=1, x(6)=3, y(6)=2. Lo cual es equivalente a afirmar que los vértices 3 y 4 están en el primer nivel de la jerarquía y el 3 precede al 4.

La variación del número de intersecciones de las aristas de G al intercambiar las posiciones de dos vértices contiguos del mismo nivel o columna debe evaluarse estudiando la posición de los vértices predecesores (en el nivel anterior) y los vértices

sucesores (en el nivel posterior) a los vértices intercambiados. Para ello se considera la siguiente definición.

Definición

Sea G=(V,E,n) una jerarquía de n niveles. Para cada pareja u,v de vértices, donde u precede a v en el mismo nivel i, se definen los parámetros siguientes:

$K_1(u,v)$ = Número de intersecciones de aristas que parten de u con aristas que parten de v.

$K_2(u,v)$ = Número de intersecciones de aristas que llegan a u con aristas que llegan a v.

$K(u,v) = K_1(u,v) + K_2(u,v)$.

Notar que si los dos vértices están en el nivel i=1 entonces, $K_2(u,v)=0$. y si i=n, $K_1(u,v)=0$.

Veamos el siguiente ejemplo en el que se intercambian las posiciones de los vértices 7 y 8.

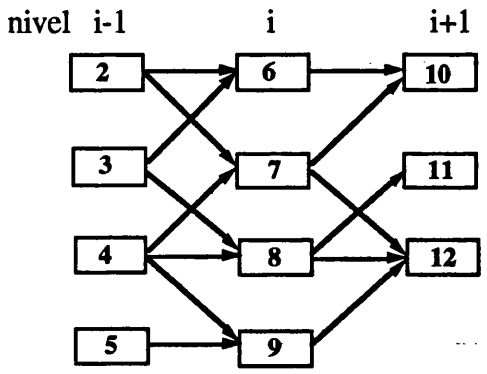


Figura 2

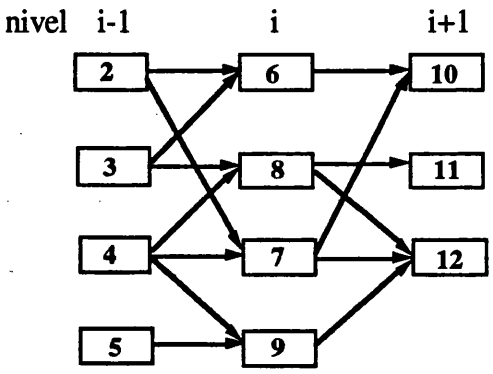


Figura 3

En la figura 2 se tiene que: $K(7,8)=K_1(7,8)+K_2(7,8)=1+1=2$, y en la 3: $K(8,7)=K_1(8,7)+K_2(8,7).=2+2=4$.

Siguiendo la clasificación del capítulo anterior respecto a la situación de dos aristas incidentes con los dos vértices a intercambiar, en jerarquías de más de dos niveles aparecen los siguientes casos:

1.- Si las aristas forman un cruce en la ordenación inicial, tras el cambio no forman cruce.
En el ejemplo ver las aristas {(3,8) (4,7)} o {(7,12) (8,11)}.

2.- Si las aristas tienen el mismo vértice final (si van del nivel i al i+1) o el mismo vértice inicial (si van del i-1 al i) no forman cruce ni en la ordenación inicial ni después del cambio.
En el ejemplo las aristas {(4,7) (4,8)}.

3.- Si las aristas no forman un cruce y no comparten ningún vértice, al intercambiar los vértices formarán un cruce.
En el ejemplo las aristas {(2,7) (3,8)}.

A partir de estas consideraciones se generaliza el algoritmo para evaluar el número de intersecciones de aristas incidentes con dos vértices u,v del mismo nivel, antes y después de permutar sus posiciones (K(u,v) y K(v,u)).

Algoritmo. K(u,v)

```
Sean los vértices u,v∈ Vi / y(u)<y(v).,
K1(u,v)=K2(u,v)=0.
K1(v,u)=K2(v,u)=0.

IF( i < n)
{
        WHILE( Existen sucesores de u sin explorar)
        {
                Sea u´en el nivel i+1 sucesor de u sin explorar.
                WHILE( Existen sucesores de v sin explorar)
                {
                        Sea v´en el nivel i+1 sucesor de v sin explorar.
                        IF ( y(u´) < y(v´) )
                                K1(v,u)=K1(v,u)+1.
                        ELSE IF ( y(u´) > y(v´) )
                                K1(u,v)=K1(u,v)+1.
                        etiquetar v´como explorado.
                }
                Etiquetar u´como explorado.
                Eliminar las etiquetas de los sucesores de v.
        }
}
```

```
IF( i >1)
{
        WHILE( Existen predecesores de u sin explorar)
        {
                Sea u´en el nivel i-1 predecesor de u sin explorar.
                WHILE( Existen predecesores de v sin explorar)
                {
                        Sea v´en el nivel i-1 predecesor de v sin explorar.
                        IF ( y(u´) < y(v´) )
                                K2(v,u)=K2(v,u)+1.
                        ELSE IF ( y(u´) > y(v´) )
                                K2(u,v)=K2(u,v)+1
                        etiquetar v´como explorado.
                }
                Etiquetar u´como explorado.
                Eliminar las etiquetas de los predecesores de v.
        }
}
K(u,v)=K1(u,v)+K2(u,v).
K(v,u)=K1(v,u)+K2(v,u).
```

2.2 DESCRIPCION DEL ALGORITMO.

Dada una jerarquía propia G=(V,E,n) con unas ordenaciones de sus niveles, se considera el conjunto ordenado M con todos los vértices de V, en donde en primer lugar están los vértices del nivel 1 con la ordenación de dicho nivel, seguidamente los del nivel 2 con su ordenación, y así sucesivamente hasta el nivel n.

Ejemplo:

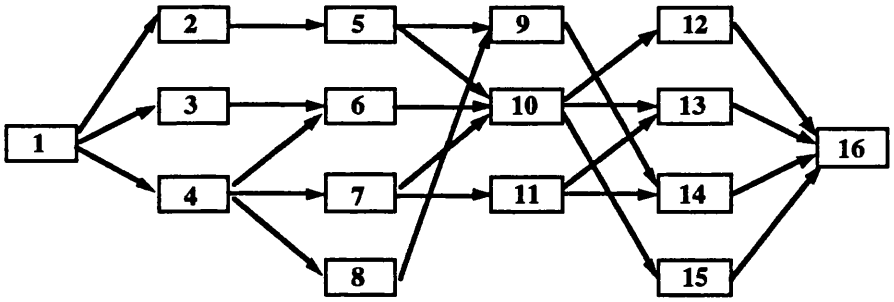


Figura 4. Ejemplo jerarquía de seis niveles

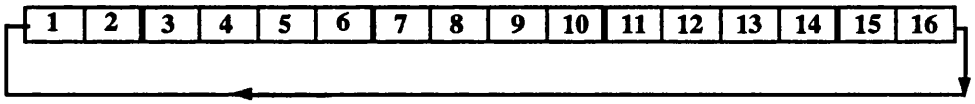


Figura 5. Lista M, del ejemplo de la figura 4

Una vez construido el conjunto M como lista circular ordenada, el algoritmo examina dicho conjunto con la estrategia Block Random Order en la fase Improving y con la estrategia Full Random Order en la fase Mixed. En cada ordenación se selecciona un vértice, se determina a qué nivel pertenece y se consideran los movimientos posibles (intercambiar dicho vértice con los adyacentes de su nivel) escogiendo el más conveniente mediante el evaluador descrito en el apartado anterior.

2.3. RESULTADOS

Para medir la bondad del algoritmo se han efectuado pruebas sobre una colección de jerarquías propias generadas aleatoriamente a partir de la densidad del grafo, el número de niveles y el número de vértices.

Para realizar las pruebas, se ha fijado la densidad de todos los grafos en 0.4 y se ha considerado que todos los niveles tienen el mismo número de vértices.

El algoritmo para generar las jerarquías aleatoriamente construye un grafo en dos pasos. En el primer paso garantiza el que todos los vértices tengan al menos grado de entrada y de salida uno (salvo los del primer y último nivel). Por lo tanto, para cada uno de los vértices de un nivel k, selecciona aleatoriamente un vértice del nivel k+1 y genera la arista correspondiente; después, para cada vértice del nivel k+1 al que no le llega ninguna arista, selecciona aleatoriamente un vértice del nivel k y genera la arista correspondiente. En el segundo paso genera aleatoriamente el resto de aristas necesarias para que el grafo tenga la densidad fijada. Para ello, en cada iteración, selecciona aleatoriamente un nivel, un vértice en dicho nivel y otro en el siguiente; si no existe la arista entre ambos, la genera.

El esquema algorítmico es el mismo que el comentado en el apartado anterior salvo que en este caso los parámetros que mejores resultados han proporcionado son: α=6, tope1=50, tope2 = 6 * |V| y tope 3=30.

La siguiente tabla muestra los resultados al aplicar el algoritmo descrito sobre una colección de ejemplos de jerarquías propias de treinta vértices distribuidos en seis capas con cinco vértices en cada capa. El valor de los parámetros es la media de tales valores sobre los quince ejemplos probados en cada clase.

Los ejemplos estudiados se han dividido en grupos según el número de niveles del grafo. Dentro de cada grupo se han distinguido diferentes subgrupos según el número de vértices. Para cada subgrupo se han considerado diez ejemplos.

Se tienen en cuenta los siguientes parámetros:

L = Número de niveles.
V = Número total de vértices.
VL = Número de vértices en cada nivel nivel.
M = Número de aristas.
INIT = Número inicial de intersecciones.
SOL. = Número de intersecciones en la solución del algoritmo.
T = Tiempo de ejecución del algoritmo (segundos salvo especificación).
T_{min} = Tiempo mínimo de ejecución en los diez ejemplos del subgrupo.
T_{max} = Tiempo máximo de ejecución en los diez ejemplos del subgrupo.

Grafos con densidad 0.4

L	V	VL	M	INIT	SOL.	T	T_{min}	T_{max}
5	20	4	26	25	4	0,4	0,4	0,6
	35	7	80	275	125	2	1,4	3,08
	50	10	160	1350	720	5	5,4	3,7
10	40	4	60	53	10	1,2	1,04	1,98
	80	8	230	1180	580	7,2	5,4	14,3
	120	12	520	6606	3923	19	16,09	26,6
15	75	5	140	240	85	3,5	2,6	5,8
	150	10	560	4630	2690	25,2	21,2	28,6
	225	15	1260	24400	16700	1m 40s	1m 8s	3m 15s
20	200	10	768	6250	3600	50	23,3	1m 9s
	400	20	3050	110590	80012	3m 40s	2m 3s	4m 50s
	600	30	6840	581586	422623	14m	12m	15m 30s

Relación con los algoritmos publicados

Eades y Kelly [9] proporcionan cuatro algoritmos para resolver el problema en grafos bipartidos. Al final del trabajo señalan que es posible generalizarlos a jerarquías de n niveles con el siguiente procedimiento:

> " Con el orden del primer nivel fijo se reordena el segundo nivel con el algoritmo. Una vez establecido el orden del segundo nivel se reordena el tercero; procediendo de esta manera hasta alcanzar el último. A continuación se repite el proceso en orden inverso, comenzando por el último nivel y terminando en el primero"

Los autores no muestran ningún resultado computacional.

El incoveniente de dicha generalización es que al estudiar en cada paso un nivel de manera aislada, no se considera la jerarquía globalmente. En grafos de dos niveles esto producía una solución que podía ser, en el mejor caso, óptima izquierda y derecha; pero no necesariamente óptima. En el caso de n niveles el algoritmo producirá resultados previsiblemente peores que en el de dos, ya que al fijar el orden de un nivel se condiciona el de todos los restantes sin ningún criterio global.

Teniendo en cuenta que para el caso de dos niveles se hizo una comparación exhaustiva entre el algoritmo tabu y los dos mejores del trabajo citado, obteniendo en todos los ejemplos probados mejores resultados con el primero, creemos que no es necesario realizar de nuevo una comparación para jerarquías de n niveles.

Warfield [53] propone un procedimiento matricial para resolver el problema de minimizar el número de cruces entre aristas de una jerarquía propia. Dicho procedimiento tiene un interés únicamente teórico, ya que por construcción, su aplicabilidad está limitada a redes de tamaños pequeños.

Sugiyama y otros [44] proponen dos algoritmos. El primero es una extensión del formulado por Warfield [53], y al igual que éste, su interés es sólo teórico. El segundo está basado en los baricentros de las filas de la matriz de interconexión, utilizando el mismo tipo de generalización descrita anteriormente en el trabajo de Eades y Kelly. Para medir la eficacia de dicho algoritmo se proporcionan unos parámetros sobre una colección de ejemplos generados aleatoriamente. El tamaño máximo de los ejemplos mostrados es de dieciséis vértices. Los autores ilustran dichos métodos mediante algunos ejemplos. En todos ellos, el algoritmo tabu proporciona mejores soluciones.

SEGUNDA PARTE

Diseño y Representación de Planos de Proyectos

CAPITULO IV

Dibujo de un Grafo Acíclico Dirigido

En este capítulo consideramos el problema de obtener una "buena" representación de un grafo acíclico con pocas intersecciones de aristas. Esto es, minimizar el número de intersecciones de aristas pero considerando en el problema restricciones introducidas para que el dibujo del grafo sea comprensible y operativo.

En el primer apartado, se estudia la asignación de los vértices del grafo a las columnas de la plantilla, transformando de este modo el grafo en una jerarquía. Para realizar dicha asignación se considera que, para respetar el criterio de que las aristas son trazadas de izquierda a derecha, el situar un vértice en una columna implica el situar sus sucesores en columnas posteriores. Con esta restricción se plantea la asignación de modo que el número de columnas de la plantilla necesarias para ubicar todos los vértices del grafo sea mínimo.

Al realizar dicha asignación, existen algunos vértices que únicamente pueden ser asignados a una columna y otros que pueden ser ubicados en varias. Considerando esta holgura, se plantea el realizar la asignación de modo que la suma de las longitudes de las aristas sea mínima[1]. Dicho problema se resuelve de manera óptima al encontrar una formulación equivalente al mismo cuyo problema dual es un problema de flujo de coste mínimo.

Una vez situados todos los vértices en columnas, y con objeto de aplicar posteriormente el algoritmo de minimización del capítulo cuarto, se sustituyen las aristas que unen vértices situados en columnas no consecutivas por cadenas de vértices y aristas ficticias, de manera que, todas las aristas resultantes unan vértices de columnas consecutivas. Al aplicar el algoritmo mencionado al grafo obtenido, los vértices quedan reordenados. De este modo, al sustituir en la solución del algoritmo las cadenas de vértices ficticios por las aristas originales pueden aparecer trazados muy irregulares.

En el segundo apartado se establecen dos criterios que inducen un "buen" trazado de una arista de longitud mayor que uno. El primero especifica que: "Todos los vértices ficticios que sustituyen a la misma arista original han de estar en la misma fila de la plantilla", y el segundo indica que: "dicha fila deberá ser cercana a las ocupadas por los vértices extremos originales de la arista".

---

[1] Se considera que la longitud de una arista es la diferencia entre número de columna de su vértice final y el inicial.

En el tercer apartado, se modifica el algoritmo heurístico del capítulo tercero de manera que, en la construcción de la solución, se considere el trazado de las aristas mediante los dos criterios establecidos anteriormente.

Con objeto de calibrar el efecto de los dos criterios de dibujo, se han comparado las soluciones proporcionadas por el algoritmo de minimización de cruces del capítulo tercero y por el algoritmo con restricciones de dibujo, sobre una colección de ejemplos generados aleatoriamente.

En el apartado 4 se reubican los vértices de cada columna sin alterar las ordenaciones obtenidas por el algoritmo de minimización con restricciones, de manera que se dibujen claramente las estructuras de ramificación y unión y se reduzca la suma total de las longitudes de las aristas[2] .

Notar que el esquema de trabajo seguido potencia el criterio de minimizar el número de intersecciones por considerarlo el más importante en el dibujo de la jerarquía. Así, se consideran otros criterios de dibujo pero de forma subordinada a este primero.

[2]En este caso la longitud de una arista es la longitud euclídea y no la simplificación considerada en el apartado 1.

1. ASIGNACION DE VERTICES A NIVELES.

En el segundo capítulo, se consideró el asignar los vértices del grafo al nivel con número más pequeño de manera que se respete la estructura de jerarquía[3]. De este modo se probó que la jerarquía así construida era mínima respecto al número de niveles. Sin embargo, dado un grafo acíclico existen otras posibilidades de asignar los vértices a niveles definiendo una jerarquía con mínimo número de niveles.

Determinemos el rango de niveles al que puede ser asignado cada vértice de manera que la jerarquía resultante sea mínima respecto al número de niveles. Para ello, se considera el grafo original y se asigna a todos los vértices una duración de 1. Aplicando los cálculos básicos del C.P.M. al grafo con dichas duraciones, el tiempo más temprano de comienzo (E.S.T.) proporciona el menor nivel al que puede ser asignado el vértice, y el tiempo más tardío de comienzo (L.S.T.) el mayor nivel. De este modo, llamaremos *vértices críticos* a aquellos en los que coinciden su menor y mayor nivel posibles y por lo tanto, han de ser asignados necesariamente a un nivel. Llamaremos *holgura de un vértice* a la diferencia entre su mayor y menor nivel de asignación.

En este apartado vamos a estudiar qué implicaciones conlleva la asignación de los vértices a determinados niveles desde el punto vista de la representación del grafo. Los siguientes ejemplos muestran diferentes posibilidades de asignar los vértices de un grafo a los niveles o columnas de una jerarquía. Ambas jerarquías representan al mismo grafo.

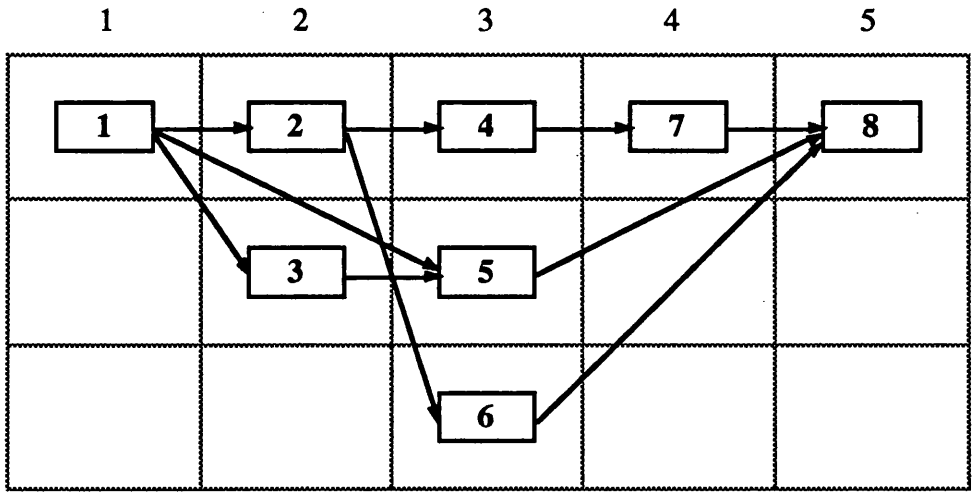


Figura 1: Jerarquía 1

[3]En toda jerarquía las aristas parten de un vértice en un nivel inferior a un vértice en un nivel superior.

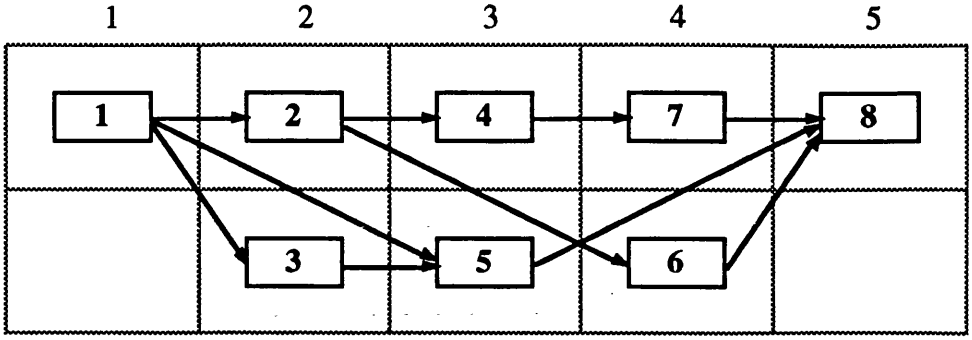


Figura 2: Jerarquía 2

La jerarquía 1 se ha construido utilizando el algoritmo del capítulo segundo. En la jerarquía 2 se ha retrasado el vértice 6 a un nivel posterior sin violar la estructura de jerarquía y sin aumentar el número de niveles. Notar que esta operación de "retraso" no se puede realizar con el vértice 2 o con el 4, puesto que retrasarlos a un nivel mayor conllevaría el aumentar el número de niveles de la jerarquía.

Como muestra el ejemplo, la asignación de los vértices no críticos a determinados niveles, tiene dos consecuencias respecto a la representación del grafo:

1.- Número de filas de la plantilla.

En el ejemplo mostrado, al situar el vértice 6 en el nivel 4, se obtiene la jerarquía 2 con una fila menos en la plantilla que la jerarquía 1. Luego parece que el número de filas de la plantilla (y por lo tanto el tamaño del dibujo) depende de la asignación de los vértices a las columnas.

2.- Longitud de las aristas.

En el ejemplo anterior, dado que el vértice 8 está fijo en el nivel 5, al situar el vértice 6 en el nivel 3 o el 4, se tiene que, la longitud de la arista (6,8) varía de 2 a 1[4]. Así pues, al situar un vértice en un nivel, se está determinando la longitud de las aristas incidentes con dicho vértice.

Estudiemos ambas cuestiones en profundidad:

[4] En el capítulo de conceptos previos, se definió la longitud de una arista como la diferencia entre el número del nivel del vértice final y el número del nivel del vértice inicial.

1.1 NUMERO DE FILAS DE LA PLANTILLA.

Tanto el algoritmo del capítulo segundo como el presentado en este capítulo, proporcionan el mínimo número de columnas en las que el grafo acíclico puede transformarse en jerarquía. Como muestra el ejemplo, el número de filas depende de la asignación que se haga de los vértices no críticos, puesto que el número de filas de la plantilla viene determinado por el máximo número de vértices en un nivel: máximo($|V_1|$, $|V_2|$, ..., $|V_n|$).

Sin embargo, dado que los algoritmos presentados trabajan con jerarquías propias, hay que transformar la jerarquía en propia. Para ello, como ya vimos, basta con sustituir las aristas de longitud mayor que uno, por cadenas de vértices y aristas ficticios. En dicho proceso de sustitución, el número de filas de la jerarquía crece, asignando a cada nivel tantos vértices ficticios como aristas lo cruzan. Por ello el número total de vértices (originales y ficticios) asignados a un nivel en la jerarquía propia, es independiente del nivel al que han sido asignados los vértices no críticos.

Así, los ejemplos siguientes, muestran el resultado de hacer propias las jerarquías de las figuras 1 y 2 presentadas en el ejemplo anterior.

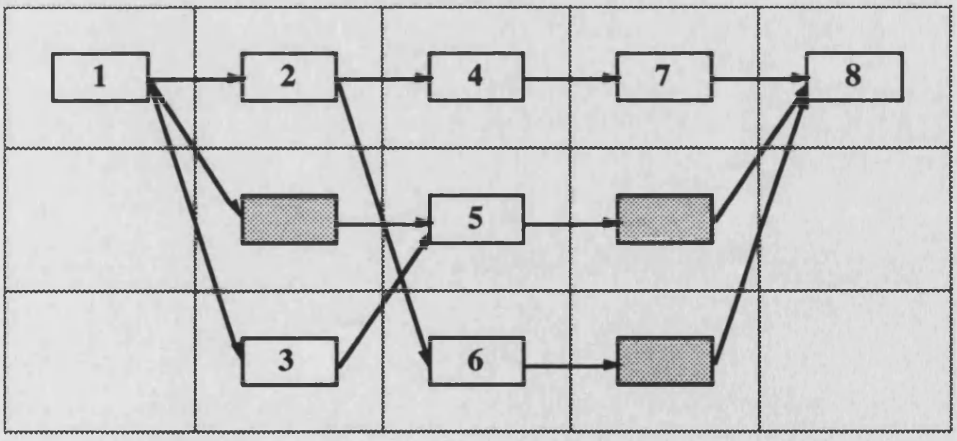


Figura 3: Jerarquía propia 1

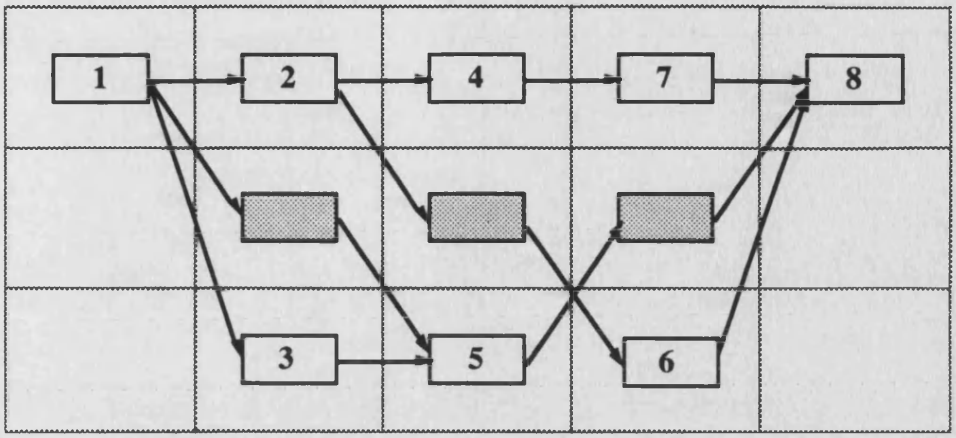


Figura 4: Jerarquía propia 2

Así pues, la asignación de los vértices a niveles no influye en el número de filas que ocupa la jerarquía al hacerla propia. Por ello, no se considerará este criterio al realizar la asignación inicial de vértices a niveles.

1.2 LONGITUD DE LAS ARISTAS.

Se considera el grafo acíclico G=(V,E) donde |V|=n y donde los vértices están ordenados topológicamente. Supongamos, sin pérdida de generalidad, que el único vértice con grado de entrada 0 es el 1 y el único vértice con grado de salida 0 es el n.

Algunos autores, como Eades y Wormald [13], señalan que el minimizar la suma de las longitudes de las aristas de un grafo es uno de los criterios que introducen claridad y facilitan la comprensión en el dibujo o representación del grafo.

La definición de longitud de una arista dada anteriormente, es una simplificación de la longitud real, ya que considera que todas las aristas entre dos niveles consecutivos tienen la misma longitud e igual a 1. Asumiendo dicha simplificación, la longitud de las aristas únicamente depende de los niveles a los que se asignan sus vértices extremos. Por otro lado, es evidente que, en orden a minimizar la longitud de las aristas, interesará representar el grafo mediante una jerarquía con mínimo número de niveles k (e igual a la longitud de un 1-n camino más largo de G).

Todo 1-n camino más largo de G tiene k vértices. El j-ésimo vértice de cada uno de dichos caminos ha de ser asignado al nivel j, ya que, de asignarlo a un nivel menor se violaría la estructura de jerarquía y asignarlo a un nivel mayor haría aumentar el número de total de niveles k. Por ello, todas las aristas en dichos caminos tienen longitud 1. Los vértices que no pertenecen a ningún 1-n camino más largo son los que tienen cierta holgura de asignación.

Notar que es equivalente hablar de longitud de una arista que del número de vértices ficticios más 1 que sustituirán a dicha arista al hacer la jerarquía propia.

Por otro lado, al implementar los algoritmos, el tamaño de las estructuras de datos que representan a un grafo es proporcional al tamaño del propio grafo, por lo que, el construir una jerarquía con menor número de vértices redundará en tiempos de computación mas bajos, proporcionando más eficiencia al algoritmo.

Por todas las consideraciones hechas, se plantea el siguiente problema.

Problema:

Dado un grafo acíclico, obtener la jerarquía con mínimo número de niveles que lo representa en la que la suma total de las longitudes de las aristas sea mínima.

Sea L igual a la suma de las longitudes de todas las aristas del grafo. Consideremos el vértice v del siguiente ejemplo.

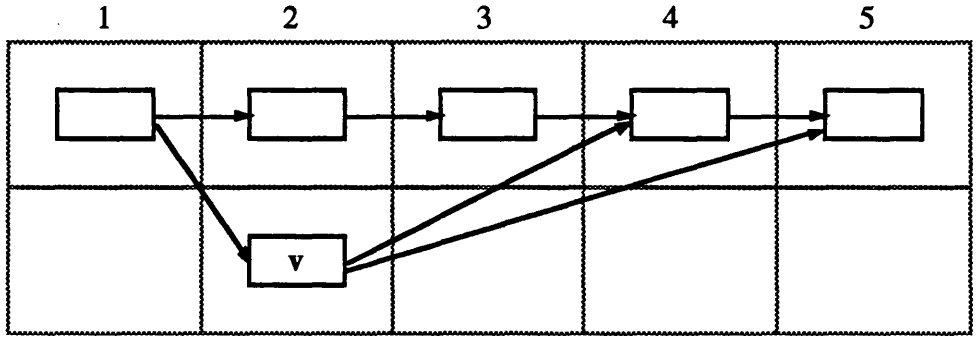


Figura 5

El vértice v puede situarse en el nivel 2 o en el 3. Sea L_1 la suma de las longitudes de todas las aristas del grafo en la distribución del ejemplo: $L_1=10$. Es claro que, por cada

nivel a la derecha que se mueva v, disminuye una unidad la longitud de cada arista que lo une con sus sucesores y aumenta una unidad la longitud de cada arista que lo une con sus predecesores. Luego definiendo t(v)=| suc(v) | - | pred(v) |=2-1=1, se tiene que, al mover v un nivel a la derecha, el valor L de la jerarquía resultante será L_1 - t(v) = 10-1=9. Además se tiene que:

- Si t(v) > 0 interesará que v esté lo más a la derecha posible: nivel 3
- Si t(v) < 0 interesará que v esté lo más a la izquierda posible: nivel 2
- Si t(v)=0 es indiferente que v esté en el nivel 2 o en el 3.

Sin embargo, en dicho análisis, no se han contemplado las posibles interferencias del vértice v con otros vértices que tengan holgura de asignación. El siguiente ejemplo muestra un caso de interferencias entre vértices. La distribución dada sitúa cada vértice en el menor nivel posible.

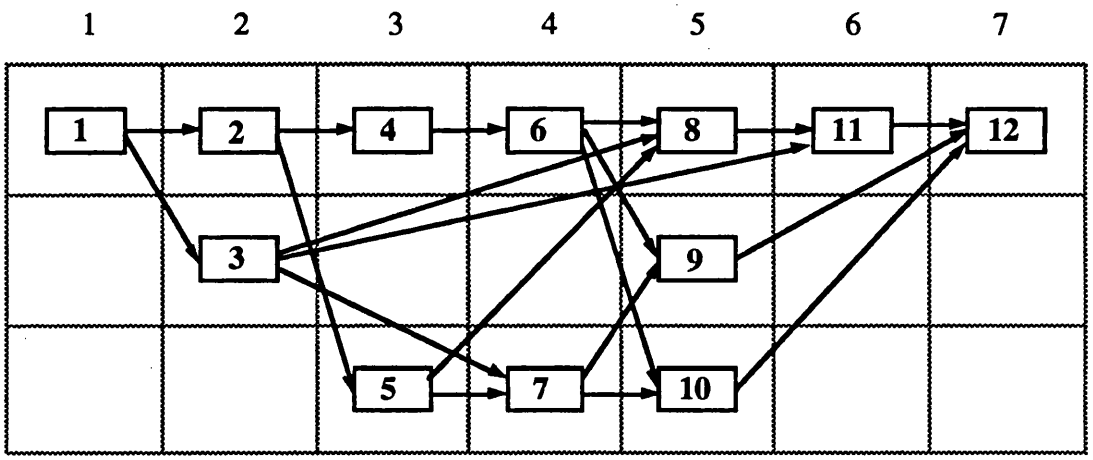


Figura 6

El vértice 3 tiene t(3)=2, por lo que interesará situarlo en el mayor nivel posible (nivel 4). El vértice 9 tiene t(9)=-1 por lo que interesará situarlo en el menor nivel posible (nivel 5). Sin embargo, dado que 7 es sucesor de 3 y predecesor de 9, las asignaciones propuestas son incompatibles, por lo que habrá que encontrar la asignación óptima para 3 y 9 de manera conjunta. Por otro lado, dado que 3 y 5 son predecesores de 7 y t(3)=2>0 y t(5)=1>0, interesará considerar de manera conjunta el desplazamiento de los 2 vértices hacia la derecha en la medida en que esto compense el desplazar los vértices 9 y 10 también hacia la derecha (t(9)=t(10)=-1<0).

Para resolver de manera óptima el problema planteado, se considera la siguiente formulación como un problema de programación lineal entera.

Para cada vértice v, se consideran los siguientes parámetros y variables:

nc(v) = Número del nivel en el que se sitúa el vértice v.

$c_1(v)$ = Menor nivel que puede ocupar el vértice v respetando la estructura de jerarquía.

$c_2(v)$ = Mayor nivel que puede ocupar el vértice v sin aumentar el número de niveles k.

Luego se tiene que: $c_1(v) \leq nc(v) \leq c_2(v)$.

Al introducir este apartado, se indicó cómo obtener los valores $c_1(v)$ y $c_2(v)$ de cada vértice del grafo.

Sea L la suma de las longitudes de las aristas en la solución actual. Sea L_1 la suma de las longitudes de todas las aristas en la distribución en la que todos los vértices están en el menor nivel posible.

Si un vértice v se mueve un nivel a la derecha se obtiene una distribución donde $L = L_1 - t(v)$. Análogamente, si v se mueve dos niveles a la derecha se obtiene una distribución con $L = L_1 - 2t(v)$. Luego la expresión que relaciona L con la posición del vértice v es:

$$L = L_1 - t(v)*(nc(v)-c_1(v))$$

Considerando que cualquier vértice puede moverse dentro de su rango, se obtiene la función objetivo:

$$L = L_1 - \sum_{v \in V} t(v)*(nc(v)-c_1(v))$$

Por otro lado hay que considerar en el problema que el nivel de un vértice ha de ser menor estrictamente que el de sus sucesores, por lo que se añade la restricción:

$$nc(v) < nc(w) \qquad \forall (v,w) \in E$$

Para que la asignación de vértices a niveles sea tal que el número total de niveles sea mínimo, se añaden las restricciones

$$nc(1) = 1 \qquad nc(n) = k$$

En donde se considera que el grafo sólo tiene el vértice 1 con grado de entrada 0 y el n con grado de salida 0. Asimismo k es la longitud de un camino más largo del grafo.

De esta manera la formulación del problema es:

$$\mathbf{P_1} \qquad \text{Minimizar} \quad L_1 - \sum_{v \in V} t(v)*(nc(v)-c_1(v))$$

s.a.:

$$nc(v) < nc(w) \qquad \forall\ (v,w) \in E$$
$$nc(1) = 1$$
$$nc(n) = k$$
$$nc(v) \in Z \qquad \forall\ v \in V$$

Considerando que L_1, $t(v)$ y $c_1(v)$ son constantes y que, en el problema original, es es equivalente el par de restricciones $nc(1) = 1$, $nc(n) = k$ al par $nc(1) \geq 1$, $nc(n) \leq k$, se reformula el problema de la siguiente manera:

$$\mathbf{P_2} \qquad \text{Maximizar} \quad \sum_{v=1}^{n} t(v)*nc(v)$$

s.a.:

$$nc(v) - nc(w) \leq -1 \qquad \forall\ (v,w) \in E$$
$$-nc(1) \leq -1$$
$$nc(n) \leq k$$
$$nc(v) \in Z \qquad \forall\ v \in V$$

Para resolver dicho problema, se considera la relajación lineal (P_2R) y se calcula el problema dual del problema relajado, obteniendo:

$$\mathbf{DP_2R} \qquad \text{Min} \quad \sum_{i,j} (-1)\, x_{ij} + y\, k + (-1)\, s$$

s.a.:

$$\sum_j x_{ij} - \sum_j x_{ji} = t(i) \quad \forall\, i \neq 1,n$$

$$\sum_j x_{nj} - \sum_j x_{jn} + y = t(n)$$

$$\sum_j x_{1j} - \sum_j x_{j1} - s = t(1)$$

$$x_{ij}\,,\ y,\ s \geq 0 \qquad \forall\, i,j$$

El problema obtenido se puede interpretar como un problema de flujo de coste mínimo, salvo por las variables y y s. Sin embargo, dado que $\sum t(v) = 0$, sumando todas las restricciones se obtiene que y=s, por lo que sin alterar el conjunto de soluciones posibles se puede añadir la restricción y-s = 0:

$$\mathbf{P_3} \qquad \text{Min} \quad \sum_{i,j} (-1)\, x_{ij} + k\, y + (-1)\, s$$

s.a.:

$$\sum_j x_{ij} - \sum_j x_{ji} = t(i) \quad \forall\, i \neq 1,n$$

$$\sum_j x_{nj} - \sum_j x_{jn} + y = t(n)$$

$$\sum_j x_{1j} - \sum_j x_{j1} - s = t(1)$$

$$y - s = 0$$

$$x_{ij}\,,\ y,\ s \geq 0 \qquad \forall\, i,j$$

El problema P_3 es un problema de flujo de coste mínimo sobre el grafo original G en el que se ha añadido un vértice ficticio f y 2 aristas: (n,f) y (f,1). El coste de todas las aristas de E es -1, el coste de (n,f) es k y el de (f,1) es -1. La oferta/demanda de cada vértice v de V es t(v), la de f es 0. La variable x_{ij} indica la cantidad de flujo que circula por la arista (i,j). Las variables y y s indican las cantidades que circulan por las arista (n,f) y (f,1) respectivamente.

Al ser P_3 un problema de flujo de coste mínimo, la matriz de restricciones es unimodular por lo que toda solución básica es entera. En particular, la solución primal óptima obtenida al aplicar el algoritmo primal del simplex especializado es entera. Por ser los costes enteros, la solución dual óptima proporcionada por el algoritmo también es entera. Considerando la notación de P_2 se puede formular el dual de P_3 de la siguiente forma:

$$\mathbf{DP_3} \qquad \text{Max} \quad \sum_{v=1}^{n} t(v)*nc(v)$$

s.a.:

$$nc(v) - nc(w) \leq -1 \qquad \forall\ (v,w) \in E$$
$$1 + z \leq nc(1)$$
$$nc(n) \leq k + z$$

en donde la variable z está asociada a la restricción y-s = 0.

TEOREMA 1

Para un valor de z fijo, existe una relación biunívoca entre el conjunto de soluciones posibles del problema P_2 y el conjunto de soluciones posibles enteras del problema DP_3. Además, dos soluciones asociadas por dicha relación tienen el mismo valor de la función objetivo.

Prueba

Supongamos el valor de z fijo. A partir de la solución $\{y_1, y_2, ..., y_n\}$ de P_2, se obtiene la solución $\{y_1+ z, y_2+ z, ..., y_n+ z, z\}$ que podemos comprobar que cumple las restricciones de DP_3 y es entera:

- La restricción $y_i - y_j \leq -1$ es la misma en los 2 problemas
- De $-y_1 \leq -1$ se tiene que $1 + z \leq y_1 + z$
- De $y_n \leq k$ se tiene que $y_n + z \leq k + z$

Análogamente a partir de la solución entera $\{y_1, y_2, ..., y_n, z\}$ de DP_3 se construye la solución $\{y_1- z, y_2- z, ..., y_n- z\}$ de P_2.

Respecto al valor de la función objetivo, sean las soluciones $\{y_1, y_2, ..., y_n\}$ de P_2 y $\{y_1+ z, y_2+ z, ..., y_n+ z, z\}$ de DP_3. A partir del hecho de que $\sum_{v=1}^{n} t(v) = 0$ se tiene el resultado:

$$\sum_{i=1}^{n} t(i)*(y_i + z) = \sum_{i=1}^{n} t(i)*y_i + z * \sum_{i=1}^{n} t(i) = \sum_{i=1}^{n} t(i)*y_i$$

■

Notar que la variable z se puede interpretar como el valor de traslación de la red sobre la plantilla. Así, si z=5, el vértice 1 estará en el nivel 5, el n en el k+5 y todos los demás estarán trasladados 5 unidades respecto a la solución con z=0.

Así pues, resolviendo el problema P_3 mediante un algoritmo de flujos en redes, se obtiene una solución óptima entera con variables duales enteras, que con una traslación proporciona la solución óptima de P_2 y por lo tanto la solución óptima del problema de asignación de vértices a niveles planteado en este apartado. Veamos un ejemplo:

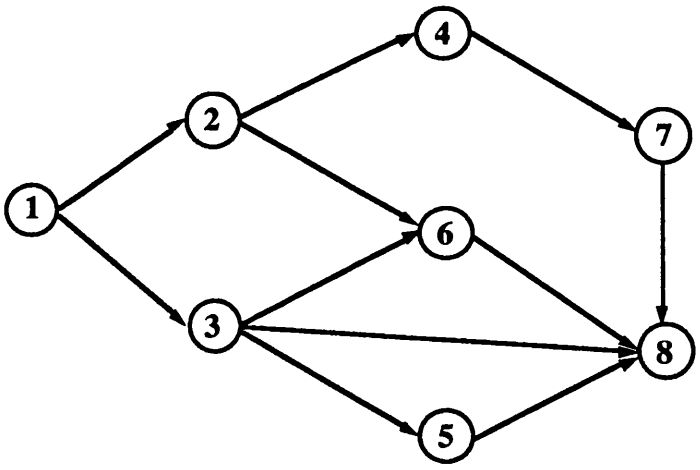


Figura 7: Grafo inicial

En el grafo de la figura 7 se tienen los siguientes valores:

vert.	c_1	c_2	t(v)
1	1	1	2
2	2	2	1
3	2	3	2
4	3	3	0

vert.	c_1	c_2	t(v)
5	3	4	0
6	3	4	-1
7	4	4	0
8	5	5	-4

Luego el problema DP_2R es un problema de flujo de coste mínimo sobre la siguiente red:

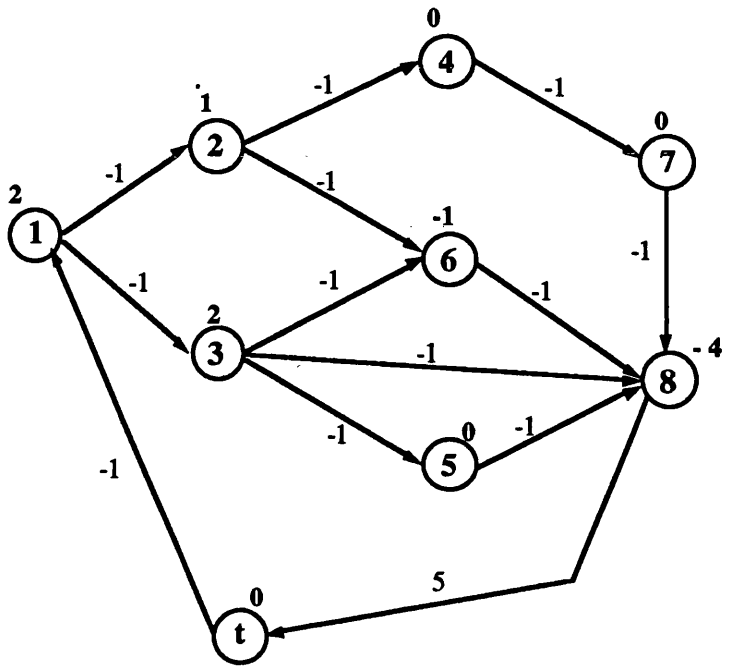


Figura 8: Grafo auxiliar

La solución óptima es: $x_{12} = 2$, $x_{24} = 3$, $x_{47} = 3$, $x_{78} = 3$, $x_{36} = 2$, $x_{68} = 1$. Con coste -14. Las variables duales son: nc(1)=4, nc(2)=5, nc(3)=nc(4)=6, nc(5)=nc(6)=nc(7)=7, nc(8)=8 y z=4. Por lo que para obtener la solución óptima del problema basta con restar z a todos los parámetros, obteniendo la distribución en niveles que muestra la siguiente figura:

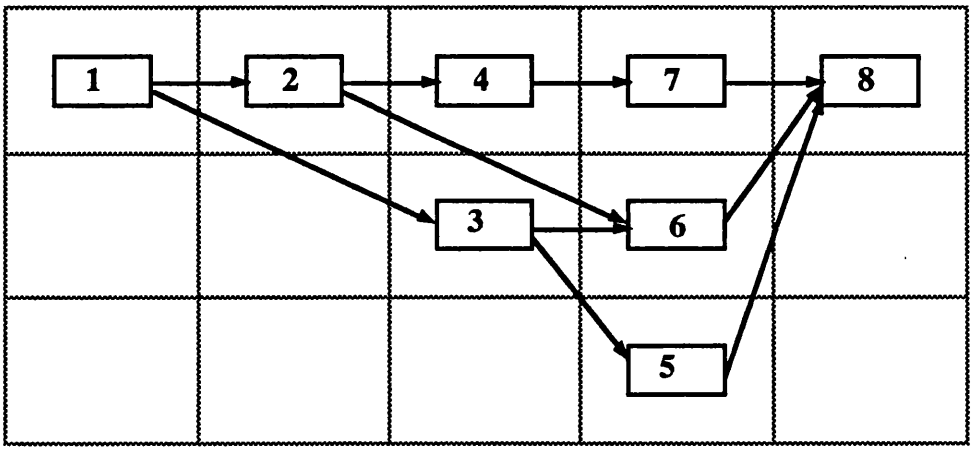


Figura 9: Grafo solución

2 CRITERIOS DE TRAZADO DE LAS ARISTAS.

Como ya se justificó en el capítulo de introducción, las aristas se dibujan mediante segmentos rectilíneos. Así, de igual manera a como lo hacen algunos autores como Sugiyama [44], Warfield [53] o Eades y Kelly [9], en el trabajo que nos ocupa, el trazado de la arista entre dos vértices de niveles consecutivos viene determinado por el segmento rectilíneo que los une. Sin embargo, dichos autores emplean el mismo tratamiento para las aristas de longitud mayor que uno dado que, una vez desdobladas, están representadas por vértices y aristas ficticias de longitud 1. A continuación se muestra porqué, a nuestro entender, las aristas de longitud mayor que 1 deben de ser estudiadas como un caso especial.

Como ya se ha comentado en varias ocasiones, para hacer la jerarquía propia se sustituyen las aristas de longitud mayor que 1 por cadenas de vértices y aristas ficticias y, una vez obtenida la solución, se restituyen las aristas originales. Veamos en un ejemplo qué problema puede aparecer en este proceso.

En la figura 10, se muestra una jerarquía impropia en donde la arista (4,13) tiene longitud dos.

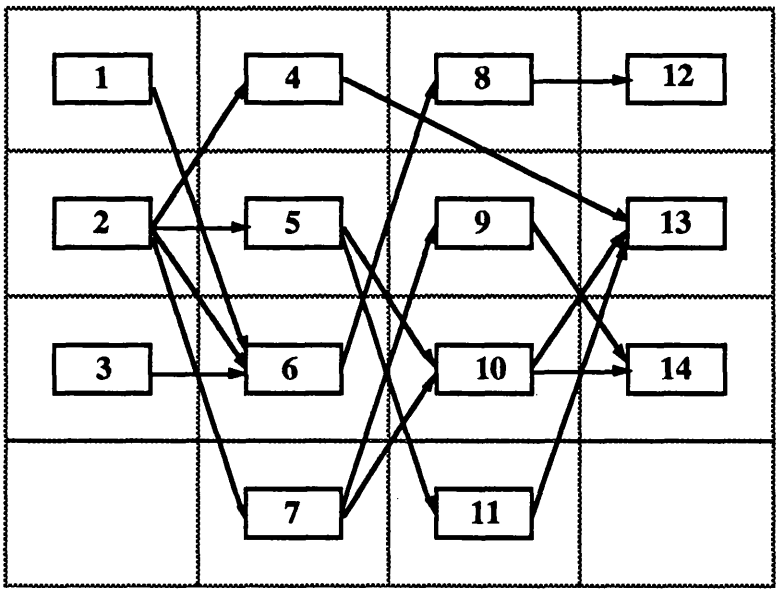


Figura 10: Jerarquía impropia original

La figura 11 muestra la jerarquía anterior, en la que se ha sustituido la arista (4,13) por dos aristas y un vértice ficticio obteniendo así la jerarquía propia.

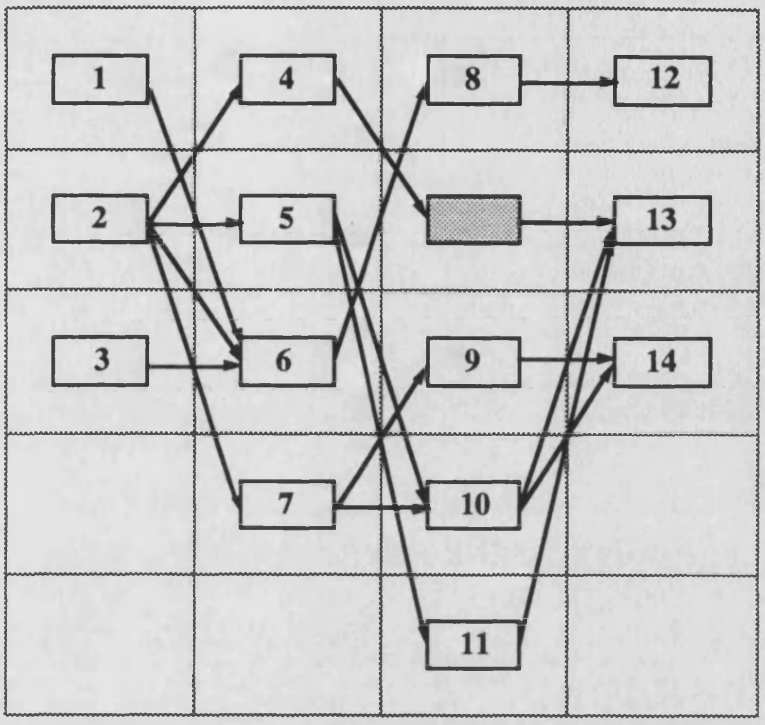


Figura 11: Jerarquía propia original

Sobre la jerarquía propia anterior se aplica el algoritmo de minimizar intersecciones visto en la parte 1 del trabajo, obteniendo la representación siguiente:

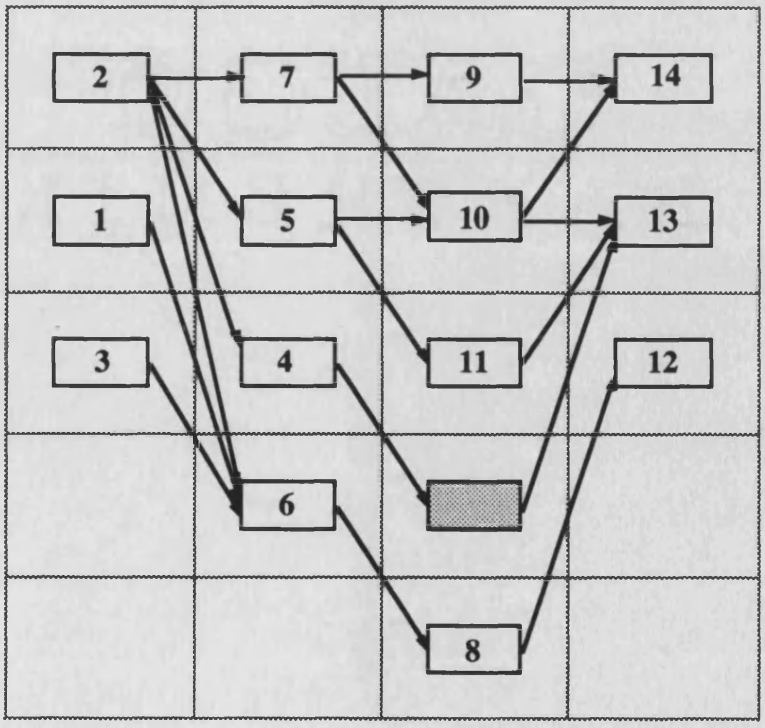


Figura 12: Jerarquía propia solución

Para obtener el grafo resultante, se sustituye, en la solución obtenida, la cadena de vértices ficticios por la arista original siguiendo el trazado de las aristas ficticias, como muestra la figura 13.

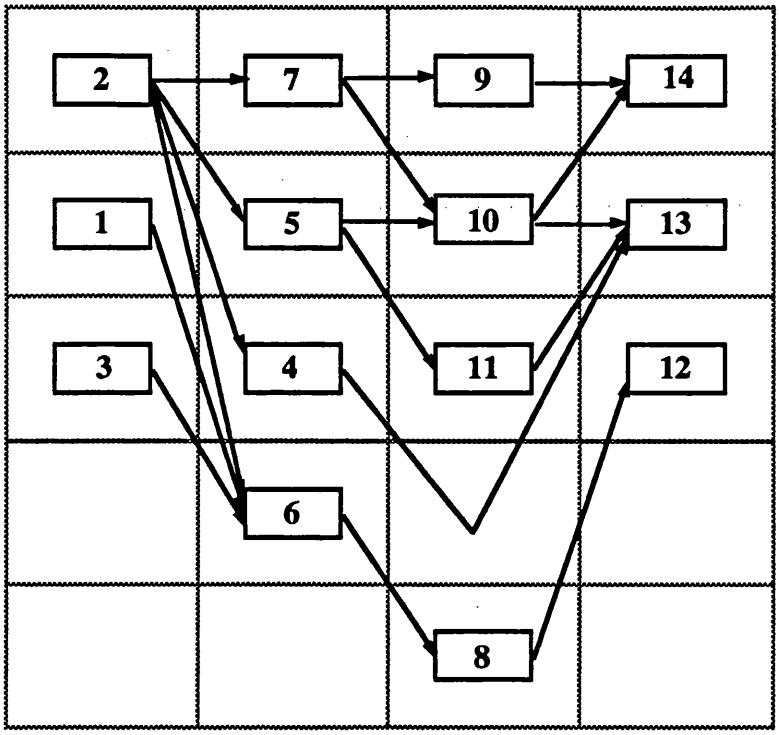


Figura 13: Grafo solución

El algoritmo para minimizar intersecciones, da el mismo tratamiento a todos los vértices: originales y ficticios. Por ello, los vértices ficticios pertenecientes a una misma cadena, pueden estar situados, en la solución óptima, en posiciones no alineadas e incluso muy alejadas. Esto provoca que al eliminar los vértices ficticios, la arista resultante tenga un trazado muy irregular.

Así en la solución final del ejemplo, el trazado de la arista (4,13) es "poco natural" considerando las posiciones que ocupan dichos vértices. Además, es evidente que a partir de este ejemplo se puede construir otro en donde los vértices 4 y 13 ocupen las mismas posiciones y el vértice ficticio esté en la columna 3 y la fila n, con n arbitrariamente grande.

En el ejemplo, se ha mostrado el problema de una arista de longitud 2. Sin embargo, el problema puede ser mucho más grave en la medida en que la arista sea de longitud mayor y esté representada por más vértices ficticios, puesto que al poder ocupar

cada vértice ficticio una posición lejana a los demás, el trazado de la arista final puede ser "serpenteante" y difícil de seguir. Por ello y por las razones expuestas en el capítulo de introducción acerca de la claridad y comprensión del dibujo, en este apartado se estudia el introducir restricciones en el problema que impidan la aparición de estas aristas "irregulares", aunque ello conlleve el aumentar el número de intersecciones de las aristas en la solución final.

Una forma de trazar las aristas de longitud mayor que uno podría consistir en eliminar los vértices y aristas ficticios de la solución proporcionada por el algoritmo que minimiza el número de intersecciones y unir, entonces, los vértices adyacentes por medio de segmentos rectilíneos. Así, en el ejemplo de la figura 13, bastaría con trazar la arista (4,13) mediante el segmento que une los dos vértices.

Sin embargo, estos segmentos no guardarían ninguna relación con las cadenas de vértices y aristas ficticios que los representaban por lo que el número de intersecciones resultante tampoco guardaría ninguna relación con el valor obtenido por el algoritmo de minimización. Por ello consideramos que los criterios de trazado de las aristas de longitud mayor que uno deben estar incorporados en el algoritmo de minimización.

Se considera la jerarquía situada en una plantilla con un sistema de referencia, de modo que cada vértice v tiene dos coordenadas:

x(v) = Columna o nivel en el que está v.
y(v) = Fila en la que está v.

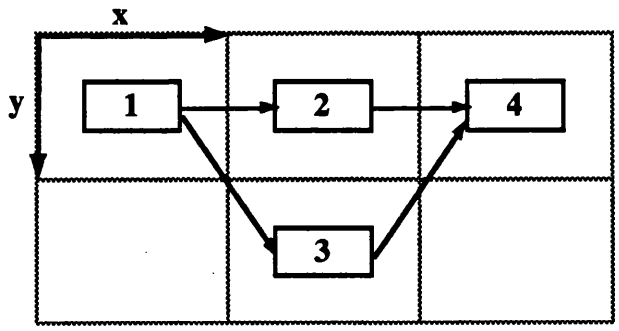


Figura 14

De este modo, en el ejemplo se tiene que: x(1)=1, y(1)=1, x(3)=2, y(3)=2, x(4)=3 y y(4)=1.

CRITERIO 1

Sea la arista (u,w) de longitud mayor que 1, de una jerarquía impropia. Sea $\{v_1, v_2, .., v_k\}$ el conjunto de vértices ficticios que sustituyen a (u,w) en la jerarquía propia mediante las aristas: (u,v_1) (v_1,v_2) ... (v_{k-1}, v_k) (v_k, w). Entonces toda solución ha de cumplir que:

$$y(v_1) = y(v_2) = ... = y(v_k)$$

es decir, todos los vértices ficticios han de estar en la misma fila.

Consideremos el problema mostrado en el ejemplo de la figura 12, en el que los dos vértices adyacentes ocupan filas cercanas pero el vértice ficticio está en una fila alejada. Este problema también puede darse en cadenas de vértices ficticios, ya que, aunque la restricción obliga a que todos los vértices de la cadena estén en la misma fila, pueden estar alejados de los vértices adyacentes originales. Por ello se considerará la segunda restricción.

CRITERIO 2

Sea la arista (u,w) de longitud mayor que 1, de una jerarquía impropia. Sea $\{v_1, v_2, .., v_k\}$ el conjunto de vértices ficticios que sustituyen a (u,w) en la jerarquía propia mediante las aristas: (u,v_1) (v_1,v_2) ... (v_{k-1}, v_k) (v_k, w). Entonces toda solución ha de cumplir que los valores:

$$\frac{y(u) + y(w)}{2} \quad \text{e} \quad y(v_j) \text{ sean cercanos.}$$

En la implementación de esta segunda restricción, habrá que precisar el hecho de que ambos valores sean cercanos.

Considerando el criterio 1, se tiene que, el trazado de las aristas de longitud mayor que uno es trivial, una vez obtenida la distribución final de vértices originales y ficticios. Así, toda arista de este tipo tiene tres tramos tal y como muestra el esquema siguiente:

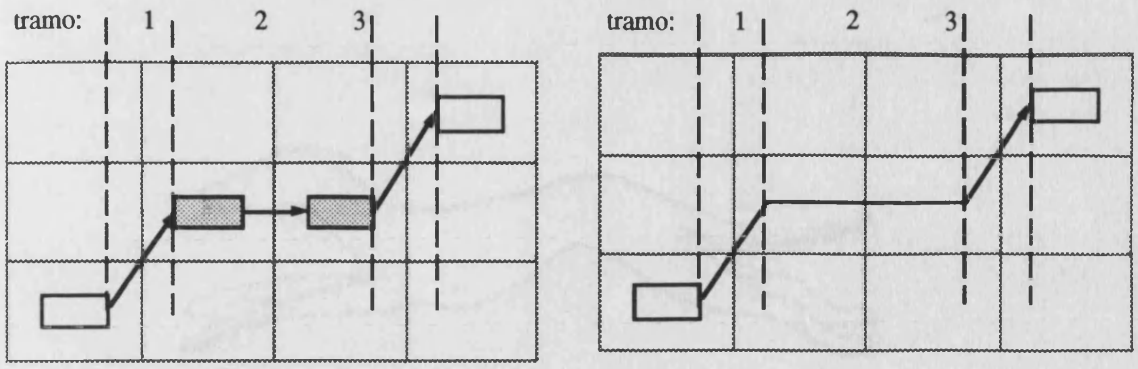


Figura 15: Trazado de una arista ficticia.

El primer tramo abarca desde el vértice original inicial hasta el primer vértice ficticio. Por estar ambos vértices en niveles consecutivos, el trazado de dicho tramo de la arista es el segmento rectilíneo que une los dos vértices y por lo tanto es trivial.

El segundo tramo comprende todos los vértices ficticios que, por encontrarse en la misma fila, son sustituidos por una linea recta, por lo que su trazado también es trivial. El tercer tramo va desde el último vértice ficticio al vértice original final y es similar al primero.

De este modo al transformar el grafo original en una jerarquía propia e incluir la restricción 1 en el problema, el trazado de todas las aristas es trivial una vez situados los vértices (originales y ficticios).

3 MINIMIZACION RESTRINGIDA DEL NUMERO DE INTERSECCIONES

El esquema algorítmico que se ha utilizado es el denominado Tabu Thresholding, dado que proporcionó excelentes resultados al resolver el problema de minimizar el número de intersecciones de las aristas y ahora vamos a resolver el mismo problema sujeto a restricciones.

El algoritmo que se va a describir parte de una solución que cumple el criterio 1. En dicho esquema, una vez seleccionado un vértice con la estrategia **Block-Random** en la *fase Improving* o con la estrategia **Full-Random** en la *fase Mixed*, se determina a qué columna pertenece el vértice y se considera el conjunto de movimientos posibles, evaluando el más conveniente según en qué fase esté el algoritmo.

3.1 DEFINICION DEL CONJUNTO DE MOVIMIENTOS.

La forma más natural para que en el grafo final todos los vértices ficticios de una cadena tengan la misma ordenada, es considerarlos de manera conjunta; es decir, una vez escogido un vértice, si es ficticio, el cambiarlo de posición supondrá automáticamente el cambiar todos los ficticios relacionados con él para que tras el cambio todos ellos sigan teniendo el mismo valor de ordenada.

En el algoritmo del capítulo anterior el conjunto de movimientos posibles únicamente depende de si el vértice está en medio de la columna o en los extremos; sin embargo al introducir la restricción sobre las cadenas de vértices y aristas ficticios se complica bastante la casuística posible. A continuación veamos en un ejemplo algunos de los casos que pueden aparecer:

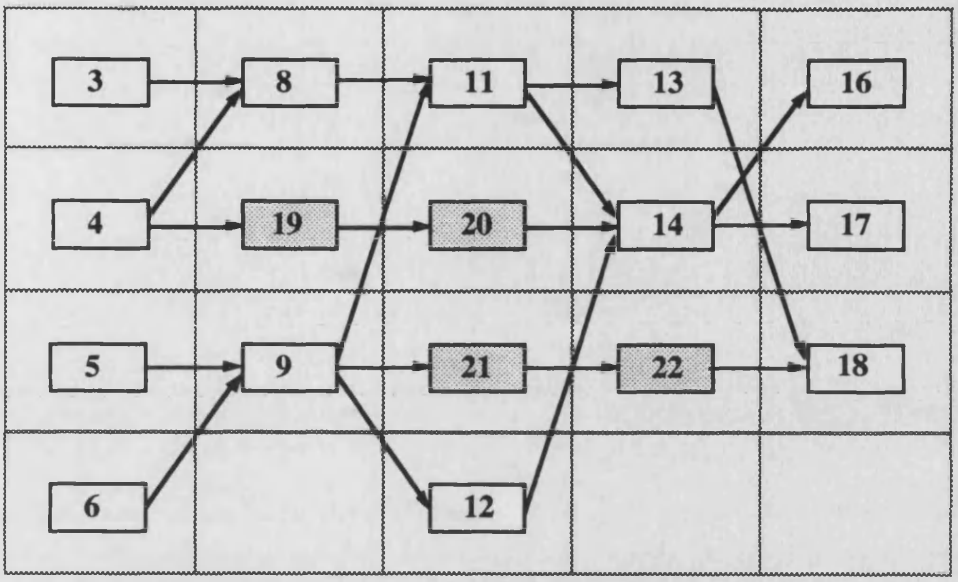


Figura 16: Jerarquía propia con vértices ficticios.

Los vértices punteados: 19,20,21 y 22 son ficticios. El ejemplo muestra como al tomar el vértice 8 e intentar permutarlo con el 19, dado que el 19 es ficticio y forma parte de la cadena {19,20}, esta permutación sólo puede realizarse si a la vez se permutan el 20 y el 11, para que el 19 y el 20 continúen estando en la misma fila.

Por otro lado, si se selecciona el vértice 9 y se decide permutar con el 19. Dado que el 19 pertenece a la cadena {19, 20} esto obligará a permutar toda la cadena, sin embargo al permutar el 20 con el 21 dado que el 21 pertenece a la cadena {21, 22} también habrá que permutar toda la segunda cadena, por lo que el resultado final, si se decide permutar el conjunto, es que los vértices 19,20 y 14 son intercambiados con 9,21 y 22 respectivamente.

Si en el ejemplo los vértices 17 y 18 también fueran ficticios, por el mismo razonamiento empleado se debería considerar la cadena de vértices ficticios que comienza en el 17 a la hora de permutar el vértice 9.

Así pues, habrá que diseñar una rutina para que, dados dos vértices donde al menos uno de ellos es ficticio, determine el rango de columnas en las que los vértices necesariamente serán permutados al permutar este vértice original. Dada la naturaleza de dicho problema se propone una función recursiva con las siguientes variables:

v,w = vértices elegidos para realizar una permutación.

y(v), y(w) = ordenadas de los vértices (número de fila).
capa_ini = número de columna a partir del cual se deben permutar los vértices.
capa_fin = número de columna hasta el que se deben permutar los vértices.
fic = detecta si se ha encontrado o no un vértice ficticio.

Cuando se llama por primera vez a esta función, capa_ini y capa_fin tienen el valor del número de columna de v y w; cont es igual a 1. La función explora alternativamente los vértices con valor de ordenada igual a y(v) e y(w). Cuando detecta un vértice ficticio, amplía el rango de columnas [capa_ini , capa_fin] si la cadena de vértices ficticios encontrada tiene vértices en columnas fuera del rango actual. La función se detiene tras explorar ambas filas en el rango ficticio [capa_ini , capa_fin] y no encontrar ningún vértice ficticio nuevo.

Algoritmo 1. Función recursiva - Rango ficticio.

```
rango_ficticio(y(v),y(w),capa_ini,capa_fin,cont)

    IF(cont es par)
        r = y(v)
    ELSE
        r = y(w)
    i = capa_ini
    fic = 0
    WHILE( i ≤ capa_fin)
        Sea u el vértice con ordenada r y abcisa i.
        IF(Existe tal vértice y es ficticio)
            Sean min_cad y max_cad el menor y mayor número de capa ocupados
            por la cadena de vértices ficticios a la que pertenece u.
            IF (min_cad < capa_ini)
                capa_ini = min_cad
                fic = 1
            IF (max_cad > capa_fin)
                capa_fin = max_cad
                fic = 1
    IF (fic = 1 o cont = 1)
        cont = cont + 1;
        rango_ficticio(y(v),y(w),capa_ini,capa_fin,cont)
```

Veamos como actúa la función recursiva rango_ficticio. Si en el ejemplo anterior se escoge el vértice 9 y se decide permutar con el 19, dado que el 19 es ficticio, se utilizará la función. Como y(9)=3, y(19)=2, x(9)=x(19)=2, la primera llamada a la función será:

rango_ficticio(2, 3, 2, 2, 1)

Dado que cont=1 es impar, se tiene que r=3. Además i=2 y fic=0. Como el vértice u en la fila r=3 y columna i=2 es el 9 no ficticio, fic sigue valiendo 0. Para prevenir esta situación en la que el vértice elegido es el no ficticio, se introduce el que si cont=1 se vuelve a llamar a la función con cont=2. Así la segunda llamada es:

rango_ficticio(2, 3, 2, 2, 2)

Ahora, dado que cont=2 es par, se tiene que r=2, i=2 y fic=0. El vértice u en la fila r=2 y columna i =2 es el 19; como es ficticio y forma parte de la cadena {19, 20} hace que capa_fin=3 y fic=1. En la siguiente iteración del bucle WHILE, i es igual a 3 y el vértice u es el 20. Notar que en cada llamada de la función los vértices que se examinan son los de una misma fila, cambiando alternativamente de fila en cada llamada.

El vértice 20 no amplía el rango ficticio. Dado que i ya es igual a capa_fin y fic=1 se vuelve a llamar a la función con los parámetros:

rango_ficticio(2, 3, 2, 3, 3)

El proceso se repite ahora con los vértices 21 y 22, terminando el algoritmo en la siguiente llamada al ser el vértice 14 no ficticio. Luego el resultado obtenido es que para permutar el vértice 9 con el 19 hay que permutar todos los vértices de las filas 2 y 3 entre las columnas 2 y 4.

Dado un vértice de la jerarquía, se define el conjunto de movimientos asociados a v, que se denota por Move(v), de manera análoga a como se hizo en el capítulo tercero.

El conjunto de movimientos Move(v) queda definido, según la posición que ocupe en su nivel, de la siguiente manera:

a) El vértice v es el primero de la ordenación de su nivel.

Sea w el siguiente vértice a v en su mismo nivel.
Si v y w son no ficticios, el único movimiento que se considera es permutar ambos vértices.
Si alguno de ellos es ficticio, el movimiento que se considera es la permutación de todos los vértices en el rango de columnas calculado con la función del algoritmo 1.

b) El vértice v es el último de la ordenación de su nivel.

Sea w el vértice anterior a v en su mismo nivel.
Si v y w son no ficticios, el único movimiento que se considera es permutar ambos vértices.
Si alguno de ellos es ficticio, el movimiento que se considera es la permutación de todos los vértices en el rango de columnas calculado con la función del algoritmo 1.

c) El vértice v no es ni el primero ni el último de la ordenación de su nivel.

Sea a el vértice anterior y p el vértice posterior a v en su mismo nivel.
Se consideran dos posibles movimientos:
Si a y v son no ficticios, el primer movimiento es permutar ambos vértices. Si alguno de ellos es ficticio, el movimiento que se considera es la permutación de todos los vértices en el rango de columnas calculado con la función del algoritmo 1.
Si v y p son no ficticios, el segundo movimiento es permutar ambos vértices. Si alguno de ellos es ficticio, el movimiento que se considera es la permutación de todos los vértices en el rango de columnas calculado con la función del algoritmo 1.

3.2 EVALUACION DE LOS MOVIMIENTOS

Dado un grafo G=(V,E) donde los vértices se han asignado a niveles definiendo una jerarquía impropia, se considerará la siguiente notación:

E_{am} = Conjunto de aristas de longitud mayor que 1.
F(v,w) = Conjunto de vértices ficticios que representa a la arista $(v,w) \in E_{am}$ al hacer la jerarquía propia.

Para medir la bondad de los movimientos asociados a un vértice, se consideran los dos evaluadores siguientes:

1.- Variación en el número de intersecciones de las aristas del grafo al aplicar el movimiento.
2.- Distancia de las cadenas de vértices ficticios F(v,w) a sus vértices adyacentes originales.

El segundo criterio únicamente se considerará si alguno de los movimientos asociados al vértice involucra a algún vértice ficticio.

3.2.1 NUMERO DE INTERSECCIONES DE LAS ARISTAS.

Como se ha visto en el apartado anterior, el introducir la restricción de que todos los vértices ficticios de una cadena han de estar en una misma fila, conlleva el que se plantee el permutar dos cadenas de vértices.

En capítulos anteriores se mostró un algoritmo para evaluar la variación en el número de intersecciones de las aristas al realizar una permutación de dos vértices. Sin embargo, a continuación se muestra un ejemplo que ilustra como, al permutar cadenas de vértices, la evaluación de la variación del número de intersecciones no se puede obtener mediante una generalización directa del algoritmo dado anteriormente.

En el siguiente ejemplo, se considera que los vértices 4 y 7 son ficticios. Por ello han de estar en la misma fila. Esto provoca el que al escoger el vértice 6 y decidir permutarlo con el 7, tengan que intercambiarse los vértices 2,6 con 4,7.

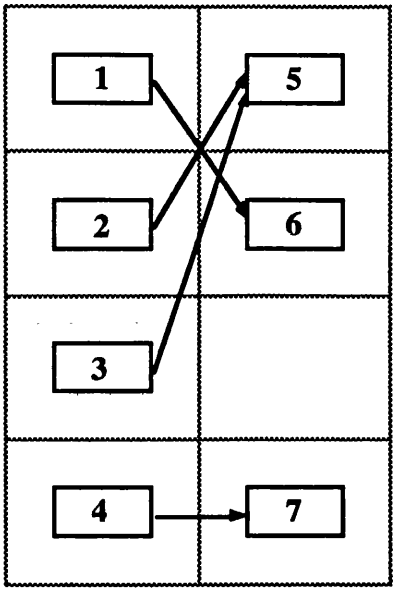


Figura 17: Ordenación 1

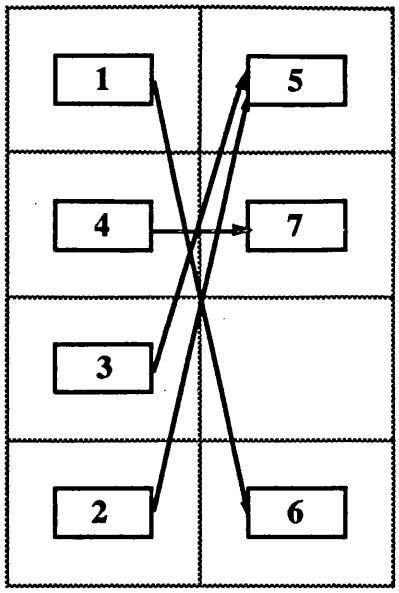


Figura 18: Ordenación 2

En la ordenación 1, el grafo tiene 2 intersecciones. Considerando que se han intercambiado el vértice 2 con el 4 y el 6 con el 7 y a partir de las expresiones dadas en capítulos anteriores, se tiene: K(4,2)-K(2,4)=1-0=1 y K(7,6)-K(6,7)=1-0=1. Esto supone un incremento total de 2 intersecciones y sin embargo la ordenación 2 tiene 5 intersecciones. Ello se debe a que en los cálculos realizados, no se ha considerado la aparición de la intersección de la arista (3,5) con la (4,7). Así pues, no basta con considerar los K(i,j) para los i,j vértices permutados ya que en el ejemplo, pese a no mover ni el vértice 3 ni el 5, generan una nueva intersección.

Por otro lado, si se añade en los grafos la arista (2,6), se tiene que ni en la ordenación 1 ni en la 2, se intersectan las aristas (4,7) y (2,6). Por ello, no sirve la expresión K(4,2)-K(2,4), ya que no se considera el caso en el que el sucesor del 4 y el sucesor del 2 son los vértices que también se permutan.

Por las razones expuestas, a continuación se estudia qué parejas de aristas cambian su situación respecto a si se intersectan o no, al realizar una permutación de dos cadenas de vértices.

Considerando las aristas (u,v) (w,t) entre dos niveles consecutivos de la jerarquía, los casos en que cambia la situación de las aristas al reordenar los vértices de ambos niveles son:

- Si en la ordenación inicial y(u)<y(w) y y(t)<y(v), las aristas se cortan y, se dejaran de cortar si al reordenar:

y(u)>y(w) y y(t)<y(v) ó y(u)<y(w) y y(t)>y(v)

- Si en la ordenación inicial y(u)<y(w) y y(t)>y(v), las aristas no se cortan y se cortarán si al reordenar:

y(u)>y(w) y y(t)<y(v) ó y(u)>y(w) y y(t)>y(v)

Así pues, la situación de las aristas cambia cuando se invierte el orden de los vértices iniciales o el de los finales, pero no ambos a la vez.

A partir de la figura de la derecha, que representa dos niveles consecutivos de vértices de una jerarquía, consideremos que los vértices a_k, b_k se permutan con los vértices a_r, b_r y estudiemos qué parejas de aristas pasan de intersectarse a no intersectarse o viceversa, al realizarse la permutación.

a_1	b_1
a_2	b_2
...	...
a_k	b_k
a_{k+1}	b_{k+1}
...	...
a_r	b_r
...	...
a_n	b_n

Caso 1.

Sean dos aristas de la forma (a_k, b_i) (a_t, b_j) con $k < t \leq r$, $1 \leq i,j \leq n$

Antes de realizar la permutación, el vértice a_k precede al a_t y tras ésta, cambia la ordenación y el a_t precede al a_k. Luego la situación de las aristas cambiará si no varía la posición relativa de b_i respecto de b_j al efectuar la permutación.

1.1 - Si i,j son distintos de k y r la permutación no afecta a los vértices b_i y b_j luego se produce un cambio respecto a la intersección de las dos aristas.

1.2 - Si i=k y, j=r (o i=r y j=k) se invierten las posiciones de b_i y b_j luego, como también se invirtieron la de los vértices iniciales de las aristas, no se produce cambio respecto a la intersección.

1.3 - Si i=k o r y, k<j<r (j=k o r y k<i<r) se invierten las posiciones de b_i y b_j luego, como también se invirtieron la de los vértices iniciales de las aristas, no se produce cambio respecto a la intersección.

1.4 - Si i=k o r y, j<k o j>r (j=k o r y i<k o i>r) no cambia la posición relativa de b_i y b_j luego se produce cambio respecto a la intersección.

Con estos cuatro casos, quedan cubiertas todas las posibilidades para b_i y b_j con $i \neq j$. Notar que si i=j es evidente que al tener las dos aristas el mismo vértice final, nunca se intersectarán.

Caso 2.

Sean dos aristas de la forma (a_t, b_i) (a_r, b_j) con $k \leq t < r$, $1 \leq i,j \leq n$

En este caso la situación es idéntica al caso anterior, ya que, antes de realizar la permutación, el vértice a_t precede al a_r y tras ésta, cambia la ordenación y el a_r precede al a_t. Luego la situación de las aristas cambiará si no varía la posición relativa de b_i respecto de b_j al efectuar la permutación. Así pues se pueden transcribir aquí los 4 subcasos anteriores cambiando únicamente el subíndice k por t y el t por r.

Notar que con estos dos casos están estudiadas todas las parejas de aristas en las que se invierte la ordenación de los vértices iniciales al realizar la permutación mencionada. Para acabar el estudio exhaustivo de todas las parejas de aristas en las que se pasa de intersección a no intersección o viceversa al realizar la permutación, faltan por estudiar aquellas en las que se produce una inversión de la situación de los vértices finales mientras que se mantiene la situación de los vértices iniciales.

Dado que en los casos 1 y 2 se han analizado tanto las situaciones en las que se produce cambio como las que no, pueden aparecer casos repetidos al estudiar las parejas en las que se permuta la ordenación de los vértices finales (esto se aprecia claramente en el caso 1.2).

Caso 3.
Sean dos aristas de la forma (a_i, b_k) (a_j, b_t) con $k < t \leq r$, $1 \leq i,j \leq n$

3.1 - Si i,j son distintos de k y r la permutación no afecta a los vértices a_i y a_j luego se produce un cambio respecto a la intersección de las dos aristas.

3.2 - Si i=k y, j=r (o i=r y j=k) este caso ya ha sido estudiado. Corresponde al caso 1.2 si t=r y al 1.3 si t<r.

3.3 - Si i=k o r y, k<j<r (j=k o r y k<i<r) ya ha sido estudiado.

3.4 - Si i=k o r y, j<k o j>r (j=k o r y i<k o i>r) no cambia la posición relativa de a_i y a_j luego se produce cambio respecto a la intersección.

Caso 4.
Sean dos aristas de la forma (a_i, b_t) (a_j, b_r) con $k \leq t < r$, $1 \leq i,j \leq n$

En este caso la situación es idéntica al caso anterior, por lo que se pueden transcribir aquí los 4 subcasos anteriores cambiando únicamente el subíndice k por t y el t por r.

Con esta clasificación se construye un algoritmo para evaluar la variación en el número de intersecciones al realizar la permutación de dos cadenas. Para ello, basta con considerar la variación entre dos capas consecutivas de la jerarquía, tal y como se ha hecho en la clasificación anterior, para después sumar las variaciones de todas las capas que ocupen las cadenas permutadas.

Sean a_k, a_r dos vértices del nivel izquierdo, y b_k, b_r dos vértices del nivel derecho, de modo que a_k, b_k están en la misma fila k, y a_r, b_r están en la misma fila r. Se considera el permutar a_k con a_r en el nivel izquierdo y, b_k con b_r en el nivel derecho. Para explorar todas las parejas de aristas, se consideran dos funciones: *Sucesores* y *Predecesores*.

La función *Sucesores* considera dos vértices del lado izquierdo que, tras la permutación, invierten su orden y estudia todas las parejas de aristas que se pueden formar con una incidente a un vértice y otra al otro. De todas estas parejas de aristas, sólo evalúa aquellas que, según la clasificación, pasan de intersectarse a no hacerlo (o

viceversa). Es decir, aquellas en las que la posición relativa de los vértices finales no varía. La función *Predecesores* realiza lo mismo salvo que considera que los vértices que invierten su posición son los del lado derecho.

En ambas funciones, el parámetro *cambio* indica el número de intersecciones nuevas que aparecerán tras efectuar la permutación. El parámetro *actual* indica el número de parejas que pasan de intersectarse a no hacerlo al efectuar la permutación. El parámetro *var*, representa la variación del número de intersecciones: *var = cambio - actual*. Así si el número de intersecciones es I antes de realizar la permutación, tras ésta el número de intersecciones será I+var = I - actual + cambio.

Algoritmo 2. Número de intersecciones

```
var = SUCESORES ( a_k , a_r , b_k , b_r )
FOR m=k+1 to m=r-1
        var = var + SUCESORES ( a_k , a_m , b_k , b_r )
        var = var + SUCESORES ( a_m , a_r , b_k , b_r )
var = var + PREDECESORES ( a_k , a_r , b_k , b_r )
FOR m=k+1 to m=r-1
        var = var + PREDECESORES ( a_k , a_r , b_k , b_m )
        var = var + PREDECESORES ( a_k , a_r , b_m , b_r )


SUCESORES (u,v,w,t)
        cambio = actual = 0
        WHILE(Existen sucesores de u sin etiquetar)
            Sea u´sucesor de u sin etiquetar.
            WHILE(Existen sucesores de v sin etiquetar)
                Sea v´sucesor de v sin etiquetar.
                IF  y(u´) ≠ y( v´)
                    IF ( (1) ó (2) ó (3) )
                        IF y(u´) < y(v´)
                            cambio = cambio + 1.
                        ELSE
                            actual = actual + 1
                Etiquetar v´ como explorado.
            Etiquetar u´ como explorado.
            Eliminar las etiquetas de los sucesores de v.
        RETURN  cambio - actual


(1) u´≠w y v´≠t y u´≠t y v´≠w
(2) ( u´=w o t )  y  ( y(v´)<y(w) o y(v´)>y(t) )
(3) ( v´=w o t )  y  ( y(u´)<y(w) o y(u´)>y(t) )
```

```
PREDECESORES (u,v,w,t)
        cambio = actual = 0
        WHILE(Existen predecesores de w sin etiquetar)
                Sea w´ predecesor de w sin etiquetar.
                WHILE(Existen predecesores de t sin etiquetar)
                        Sea t´predecesor de t sin etiquetar.
                        IF y(t´) ≠ y(w´)
                                IF ((4) ó (5) ó (6) )
                                        IF y(w´) < y(t´)
                                                cambio = cambio + 1.
                                        ELSE
                                                actual = actual + 1
                        Etiquetar t´ como explorado.
                Etiquetar w´ como explorado.
                Eliminar las etiquetas de los predecesores de w.
        RETURN  cambio - actual
```

(4) w´≠u y t´≠v y w´≠v y t´≠u
(5) (w´=u o v) y (y(t´)<y(u) o y(t´)>y(v))
(6) (t´=u o v) y (y(w´)<y(u) o y(w´)>y(v))

Veamos en el siguiente ejemplo como actúa el algoritmo en el que se intercambian las cadenas de vértices {3,8} con {5,10} :

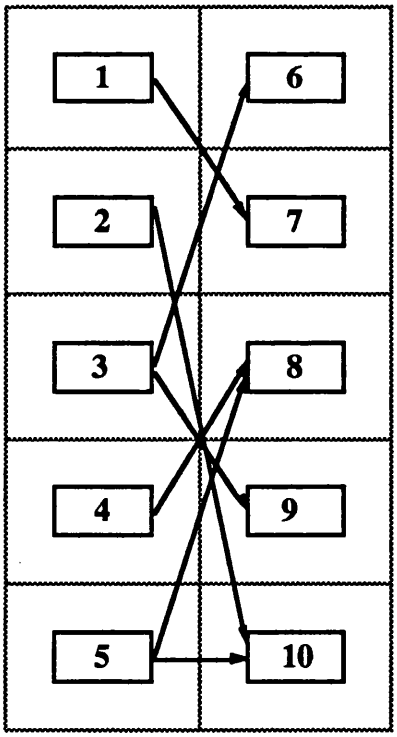


Figura 19. Ordenación inicial: 7 cortes.

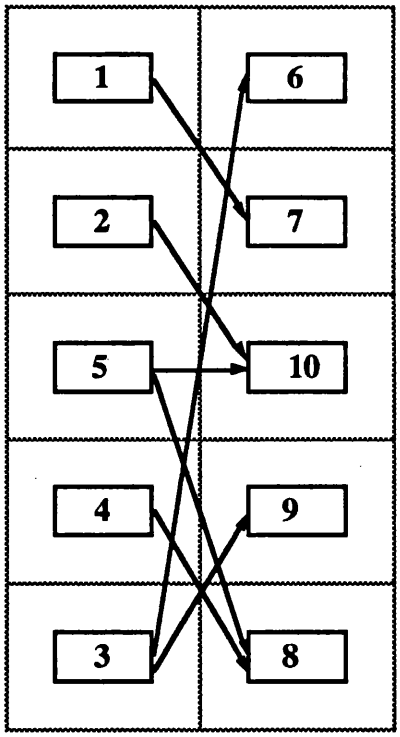


Figura 20. Ordenación final: 7 cortes.

Veamos las sucesivas llamadas a las dos funciones y cómo va evolucionando el parámetro *var*:

Sucesores(3, 5, 8, 10)	cambio=2, actual=0	var = cambio-actual = 2.
Sucesores(3, 4, 8, 10)	cambio=1, actual=0	var = var + cambio - actual = 3
Sucesores(4, 5, 8, 10)	cambio=actual=0	var = var + cambio - actual =3
Predecesores(3, 5, 8, 10)	cambio=0, actual=2	var = var + cambio - actual =1
Predecesores(3, 5, 8, 9)	cambio=actual=0	var = var + cambio - actual =1
Predecesores(3, 5, 9, 10)	cambio=0, actual =1	var = var + cambio - actual =0

Luego el número de intersecciones tras la permutación de las dos cadenas es el número original de intersecciones, que es 7, más el incremento, que es 0, por lo tanto 7.

Es evidente que si la cadena de vértices que se intercambia ocupa más niveles de la jerarquía, bastará con aplicar el procedimiento descrito para cada par de niveles consecutivos.

3.2.2. DISTANCIA DE LAS CADENAS FICTICIAS.

En el segundo apartado se introdujo el criterio 2 para el trazado de las aristas. Este criterio especifica el que la cadena de vértices ficticios no esté muy alejada de sus vértices adyacentes originales.

Para implementar tal criterio, se consideran las siguientes definiciones que permitirán evaluar aquellos movimientos en los que aparezcan vértices ficticios.

Definición

Dada la cadena de vértices ficticios F(v,w) incidente a los vértices originales v y w, se define la distancia de la cadena como[5]:

$$d(\,F(v,w)\,) = |\;E\left(\frac{y(u)+y(w)}{2}\right) - \;y(\,F(v,w)\,)\;|$$

Definición

Se define la *distancia de un movimiento* del siguiente modo: Si el movimiento consiste en el intercambio de dos vértices no ficticios, su distancia es 0. En otro caso, se define la distancia del movimiento como la media de las distancias de todas las cadenas F(v,w) de vértices ficticios que son permutadas al realizar el movimiento.

[5] E(x) indica la parte entera de x y |z| el módulo del número z.

3.3 SELECCION DEL MOVIMIENTO

Una vez seleccionado un vértice v, se calcula el conjunto de movimientos Move(v) asociado al vértice, tal y como se ha descrito en el apartado 3.1. Cada uno de los movimientos se evalúa con los dos parámetros introducidos en el punto 3.2. A continuación veamos qué movimiento se selecciona, en función de los evaluadores y de la fase en la que se encuentre el algoritmo.

El escoger un vértice ficticio y decidir permutarlo con otro vértice obliga a permutar, al menos, toda la cadena F(v,w) de vértices ficticios unidos a él. Por ello, al aplicar en el algoritmo las elecciones pseudoaleatorias de vértices, tanto en la fase Mixed como en la Improving, las aristas $(v,w) \in E_{am}$ serán movidas un número mayor de veces que las aristas de longitud uno.

Por otro lado, como se ha visto en apartados anteriores, el evaluar el cambio en el número de intersecciones al mover un vértice ficticio requiere de un esfuerzo computacional mucho mayor que con un vértice original.

Definición

Se define el conjunto de movimientos no ficticios, que se denotará por NF_Move(v), como el conjunto de movimientos de Move(v) que no involucran a vértices ficticios.

A partir de las consideraciones hechas, se establecen los siguientes criterios de selección del movimiento.

FASE IMPROVING

En la fase Improving, el criterio 2 de la distancia se implementa de la manera siguiente: sólo podrán realizarse movimientos que involucren vértices ficticios si la distancia del movimiento es menor o igual que una cantidad prefijada que llamaremos *umbral* [6].

[6] El valor del umbral se determinará computacionalmente.

Definición

Se define el conjunto de movimientos restringidos, que se denotará por Move(v)*, como el conjunto resultante de eliminar de Move(v) aquellos movimientos cuya distancia sea mayor que el umbral.

El criterio de selección del movimiento se establece de la siguiente manera:

Si NF_Move(v) $\neq \emptyset$, se toma de dicho conjunto el movimiento que más disminuya el número de intersecciones. En otro caso, si Move(v)* $\neq \emptyset$, se toma de este conjunto el movimiento que más disminuya el número de intersecciones.

Si ambos conjuntos son vacíos, no se selecciona movimiento.

FASE MIXED

Si NF_Move(v) $\neq \emptyset$, se toma de dicho conjunto el movimiento que más disminuya el número de intersecciones. En otro caso, se selecciona el movimiento de Move(v) que más disminuya su distancia.

3.4 DESCRIPCION ALGORITMICA.

El esquema global del algoritmo es el mismo que en el problema sin restricciones que se mostró en el capítulo tercero. A continuación se detallan los esquemas de las dos fases.

Algoritmo 3. Fase Improving

1.- Construir M como una lista circular ordenada, donde primero están los vértices del nivel 1 con la ordenación inicial, después los del 2 y así sucesivamente hasta el n.
no_cambio = 0.

2.- Sea I el intervalo formado por los α elementos siguientes (en la primera iteración tomar los α primeros).

3.- Seleccionar v aleatoriamente de I. Calcular Move(v), NF_Move(v) y Move(v)*.

4. - IF (Existe algún movimiento de NF_Move(v) que disminuya el número de cruces)
Realizar el movimiento de NF_Move(v) que más disminuye el número de cruces.
Actualizar el nivel x(v) y M.
no_cambio = 0.

ELSE IF (Existe algún movimiento de Move(v)* que disminuya el número de cruces)
Realizar el movimiento de Move(v)* que más disminuye el número de cruces.
Actualizar el nivel x(v) y M.
no_cambio = 0.

ELSE
No realizar movimiento.
no_cambio = no_cambio + 1.

5.- IF(no_cambio = tope1)
parar.
ELSE
ir a 2.

Algoritmo 4. Fase Mixed

1.- Considerar M con el orden resultante de la fase Improving.
n_it = 1.

2.- Selecciona v aleatoriamente de M. Calcular Move(v), NF_Move(v).

n_it = n_it + 1.

3. - IF (NF_Move(v) ≠ ∅)
Realizar el movimiento de M_nF(v) que más disminuye el número de cruces.

ELSE
Tomar el movimiento que más disminuya el parámetro distancia.

Actualizar la ordenación del nivel x(v).
Actualizar la ordenación de M.

4.- IF (Solución obtenida tiene menos cruces que la almacenada como mejor solución)
Guardar la solución actual como mejor solución.

5.- IF (n_it = tope2)
parar.
ELSE
ir a 2.

Tras realizar numerosas pruebas con diferentes ejemplos de jerarquías, se han establecido los valores: $\alpha = 6$, tope1=50, tope2=3*(|V| + |E|) y tope3=50.

El parámetro umbral puede ser definido por el usuario en cada instancia del problema en función de la holgura que se le permita a la restricción 2.

3.5 ASIGNACION INICIAL DE VERTICES A FILAS.

El siguiente algoritmo proporciona una distribución inicial de los vértices del grafo que cumple el criterio 1, considerando que los vértices originales ya están asignados a niveles.

Algoritmo 5. Asignación inicial de vértices a filas.

Primer Paso

WHILE ($E_{am} \neq \emptyset$)
Sea (v,w) $\in E_{am}$
IF(v no ha sido asignado)
Asignar v a la menor fila posible en el nivel x(v)
Eliminar v de V
IF(w no ha sido asignado)
Asignar w a la menor fila posible en el nivel x(w)
Eliminar w de V
Obtener la fila i, cumpliendo que es la menor para la que no hay vértices asignados en todas las columnas comprendidas en [x(v),x(w)]
Asignar todos los vértices de F(v,w) a la fila i y la columna correspondiente.
Eliminar (v,w) de E_{am}

Segundo Paso

WHILE ($V \neq \emptyset$)
Sea v $\in$ V
Asignar v a la columna x(v) y a la menor fila libre en dicha columna.
Eliminar v de V

Ejemplo: Consideremos el siguiente grafo acíclico ordenado topológicamente en el que los vértices 1 y 17 son ficticios representando comienzo y terminación:

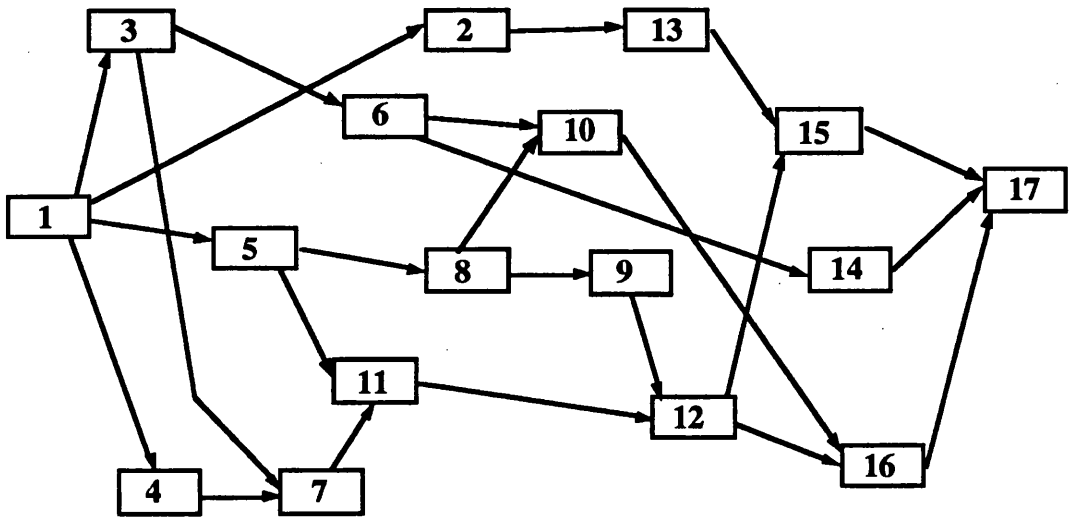


Figura 21: Grafo acíclico

En dicho grafo se calculan los números de niveles de todos los vértices con el procedimiento descrito en el apartado 1, obteniendo:

Vértice	nivel
2	1
3	1
4	1
5	1
6	2

Vértice	nivel
7	2
8	2
9	3
10	3
11	3

Vértice	nivel
12	4
13	4
14	5
15	5
16	5

En el paso 1 se obtiene la asignación de los vértices ficticios y los originales adyacentes, de la siguiente manera:

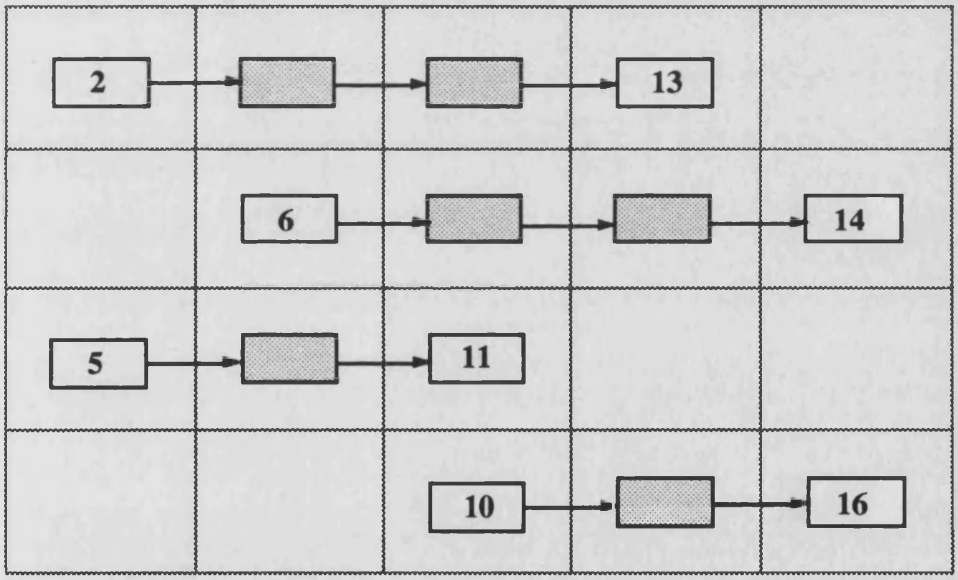


Figura 22: Asignación de vértices incidentes con aristas de E_{am}

En el paso 2 se completa la asignación de los vértices restantes, obteniendo:

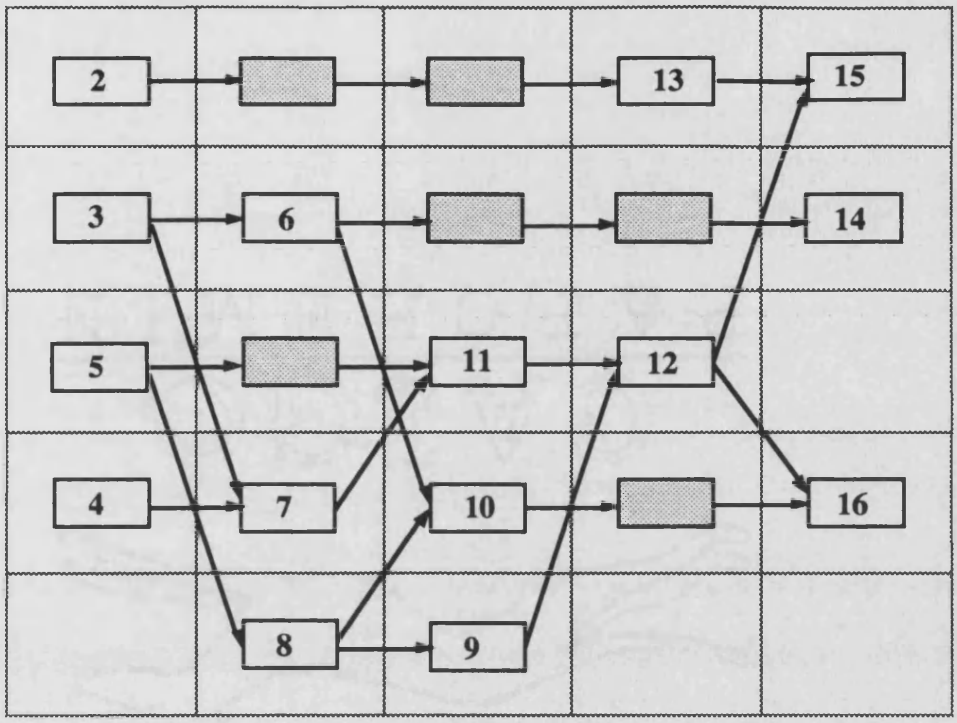


Figura 23: Asignación del resto de vértices

3.6 RESULTADOS

Para medir la eficiencia del algoritmo presentado, se han efectuado pruebas sobre una colección de ejemplos generados aleatoriamente. Las pruebas se han realizado sobre tres clases de grafos.

C1= Grafos acíclicos que al transformarse en jerarquías propias tienen 50 vértices distribuidos en 10 niveles.
C2=Grafos acíclicos que al transformarse en jerarquías propias tienen 50 vértices distribuidos en 6 niveles.
C3=Grafos acíclicos que al transformarse en jerarquías propias tienen 100 vértices distribuidos en 16 niveles.

Para contrastar los resultados se toman como referencia los resultados que se obtienen con el algoritmo heurístico de minimización de cortes presentado en el capítulo anterior[7]. Asimismo, para medir el efecto de las restricciones, se evaluará, para dicho algoritmo sin restricciones, la situación final de las cadenas de vértices y aristas ficticios.

Por otra parte, dado que el parámetro *umbral* es definido por el usuario en cada ejecución del programa, se considerarán dos posibles valores de dicho parámetro. Por las razones expuestas, se consideran los 3 algoritmos siguientes:

A0=Algoritmo de minimizar intersecciones presentado en el capítulo anterior.
A1=Algoritmo presentado en este capítulo con valor de umbral=10.
A2=Algoritmo presentado en este capítulo con valor de umbral=3.

Notar que dado el tamaño de los problemas test, asignar un valor de 10 al parámetro umbral es equivalente a eliminar el criterio 2 del modelo. Por ello, se puede considerar que el algoritmo A0 no tiene ninguna restricción de dibujo; el A1, una restricción y el A2 dos restricciones. La segunda restricción del A2 especifica que la cadena de vértices ficticios
no se puede situar a más de 3 unidades de distancia de la media de las posiciones de sus vértices originales adyacentes ($d(F(v,w))<3$).

[7]Como ya se mostró, estos resultados son muy cercanos al óptimo.

Para contrastar los algoritmos se consideran los siguientes parámetros:

c = Número de intersecciones de la solución del algoritmo.

q = Mide el número de veces que se "quiebra" una arista de longitud mayor que uno en la solución final del algoritmo A0. Es decir, para cada cadena de vértices ficticios, cuenta el número de veces que la ordenada de un vértice ficticio no coincide con la de su sucesor ficticio, sumando dicha cantidad para todas las cadenas.

s = En los algoritmos A1 y A2 es la suma de las distancias de las cadenas ficticias: d(F(v,w).

m = En los algoritmos A1 y A2 es el máximo de dichas distancias.

Considerando diez ejemplos para cada una de las clases descritas, la siguiente tabla muestra la media de cada parámetro en cada clase, así como la media sobre todas las clases.

	Origen	**A0.c**	**A0.q**	**A1.c**	**A1.s**	**A1.m**	**A2.c**	**A2.s**	**A2.m**
C1	45.16	5.3	13.6	6.1	5.16	2.5	6.3	4.6	2.16
C2	51.5	16.5	12.25	18	6	2.25	21	3.75	1.25
C3	93.2	21	21	25.25	6.25	1.75	21.25	6	1.25
MEDIA	63.2	14.2	15.6	16.45	5.8	2.2	16.2	4.8	1.5

Los algoritmos han sido programados en C y ejecutados en un ordenador personal 486-66Mhz. Los tiempos de computación del algoritmo A0 son de escasos segundos (fueron mostrados en el capítulo anterior). Los de los algoritmos A1 y A2 son superiores a estos pero siempre inferiores a un minuto.

Conclusiones.

El algoritmo A0 al no tener restricciones es el que menor número de intersecciones proporciona (A0.c). Sin embargo, al no considerarse ninguna restricción sobre las aristas y vértices ficticios, vemos cómo se producen un gran número de "quiebros" de las aristas de longitud mayor que uno.

Al comparar los parámetros de distancia máxima (m) y suma de distancias (s) en los algoritmos A1 y A2, se puede ver que, aunque el valor de los parámetros es inferior en el A2, los valores de A1 son muy cercanos a estos. Ello es debido a que, el criterio 1 induce de por si un acercamiento de la cadena F(v,w) a sus vértices adyacentes originales v y w.

Por otro lado, el número de intersecciones (c) de las soluciones proporcionadas por los algoritmos A1 y A2 es cercano a los del algoritmo A0. Sin embargo, aunque el algoritmo A2 está más restringido que el A1, en los ejemplos de tamaños grandes (C3) los resultados del algoritmo A2 son un poco mejores que los del A1 respecto al número de intersecciones. Aunque en el resto de clases no ocurre así, existe muy poca diferencia entre las soluciones de ambos algoritmos.

Los resultados mostrados en la fila MEDIA de la tabla parecen indicar que, globalmente considerado, el algoritmo A2 es superior a los otros algoritmos.

4 REDISTRIBUCION CON CRITERIOS OPERATIVOS DE DIBUJO.

En este apartado se mantiene el esquema algorítmico Tabu Thresholding utilizado en el apartado anterior.

Se toma como solución inicial la obtenida con el algoritmo descrito en el tercer apartado, que proporciona la distribución de los vértices que minimiza las intersecciones de las aristas, sujeta a las dos restricciones de dibujo.

En este apartado se muestra un algoritmo que reasigna los vértices a la plantilla de modo que no se alteran las ordenaciones obtenidas en cada nivel de la solución inicial y por lo tanto, el número de intersecciones. El objetivo de dicho algoritmo es obtener una representación más clara y operativa del grafo obtenido con el algoritmo del apartado tercero.

Los criterios que se van a aplicar a tal efecto son los siguientes:

1.- Minimizar la suma de las longitudes de las aristas.

Como ya se comentó en el capítulo de introducción, algunos autores como Tagawa [45] o Wormald [13] proponen la consideración y optimización de este criterio al representar un grafo. En este caso se considera la definición tradicional de distancia euclídea y no la simplificación que se aplicó en el apartado 1.

Notar que el minimizar la suma de las longitudes de las aristas implica el maximizar el número de aristas que se dibujan horizontalmente lo cual, tal y como se justificó en [34], es un buen criterio para representar un grafo.

2.- Equilibrado de las aristas incidentes con un vértice.

Este criterio potencia las estructuras de ramificación y unión aportando claridad al grafo, tal y como muestran los siguientes ejemplos:

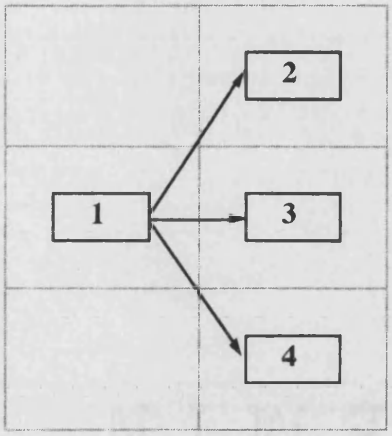


Figura 24: Ejemplo A

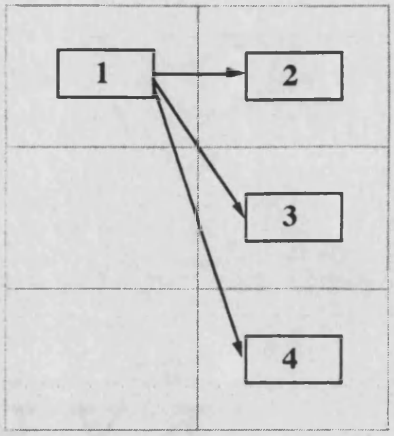


Figura 25: Ejemplo B

Notar como en el ejemplo A la situación de "centrado" del vértice 1 frente al 2,3 y 4 reafirma la relación que muestran las aristas, cosa que no ocurre en el ejemplo B; dicho de otro modo, casi no son necesarias las aristas para entender que el 1 es predecesor del 2,3 y 4.

Se considera como anteriormente que la jerarquía está situada en una plantilla o cuadrícula en donde para cada vértice v, x(v) indica la columna o nivel al que pertenece, e y(v) la fila que ocupa.

Considerando que el esquema algorítmico Tabu Thresholding está suficientemente detallado en apartados anteriores, no comentaremos aquí la descripción algorítmica, describiendo únicamente los aspectos específicos del procedimiento diseñado.

4.1 DEFINICION DEL CONJUNTO DE MOVIMIENTOS

Para todo vértice v de la jerarquía propia G=(V,E,n) se consideran dos posibles movimientos:

1.- Mover v a la casilla inferior en su columna.
2.- Mover v a la casilla superior en su columna.

Dado que no se pueden alterar las ordenaciones de las columnas, si hay un vértice en la casilla a la que se va a desplazar v, el realizar el movimiento implicará el mover también

dicho vértice. Así, en el siguiente ejemplo (Fig. 26) al tomar el vértice 2 y considerar los dos movimientos posibles, hay que tener en cuenta que, situarlo en la casilla superior obliga a desplazar al vértice 1 (Fig 27).

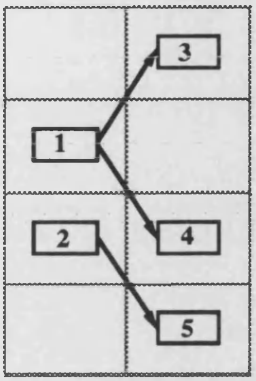


Figura 26

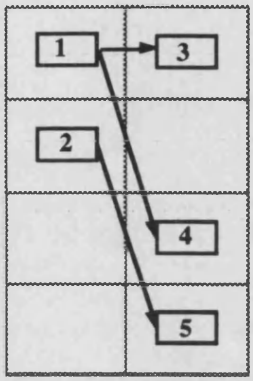


Figura 27

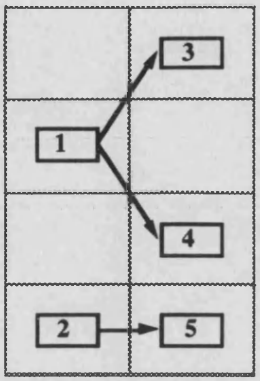


Figura 28

El mover el vértice 2 a la casilla inferior no desplazará a ningún otro vértice (Fig 28).

Veamos como las aristas de longitud mayor que uno, que en el grafo actual están representadas por vértices y aristas ficticios (F(v,w)), provocan el que, al realizar un movimiento de los mencionados, se vean alteradas varias columnas.

Consideremos el ejemplo de las figuras 29 y 30 en el que los vértices sombreados son ficticios.

Dado que, los vértices de una cadena F(v,w) han de tener el mismo valor de ordenada, al mover el vértice 1 a la casilla inferior ha de moverse el vértice 2. Como el vértice 2 pertenece a la cadena de vértices ficticios {2,5,8}, para que tengan la misma ordenada, han de moverse los vértices 5 y 8, lo cual implica el mover los vértices 3 y 6.

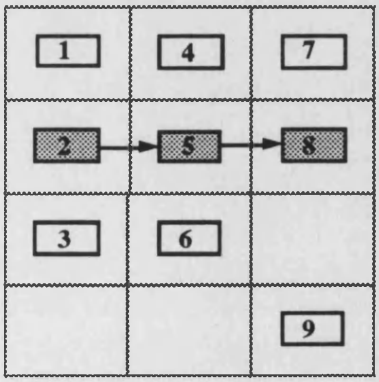


Figura 29

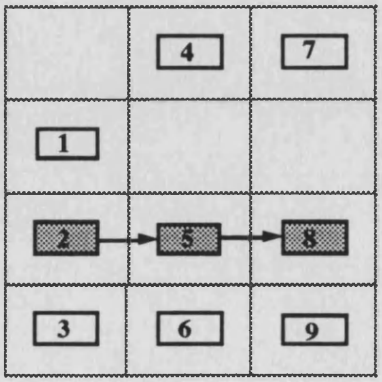


Figura 30

Para calcular el conjunto de vértices que han de moverse al mover un vértice v se propone la siguiente función recursiva.

Sea v el vértice seleccionado, sea A el conjunto de vértices que necesariamente se han de mover al mover v y sea a un parámetro que puede tomar dos valores: a=1 indica que v se va a mover a la casilla inferior y a=-1 a la superior. Como anteriormente, nc(v) indica el número de columna de v e y(v) su número de fila en la plantilla.

Se notará por [x,y] a la casilla que está en la columna x y en la fila y de la plantilla. La expresión lógica [x,y]=0 indica que en dicha casilla no hay ningún vértice. Por el contrario [x,y]=1 indica que está ocupada por un vértice.

Algoritmo 6. Función recursiva - Conjunto a mover.

```
mover(v,a)
        A = A ∪ {v}
        IF( v es ficticio)
                Sea F(v) la cadena de vértices ficticios en la que está v.
                A = A ∪ F(v)
                WHILE( F(v) ≠ ∅ )
                        Sea w ∈ F(v)
                        IF( [ nc(w),y(v)+a ] =1)
                                Sea t el vértice en tal casilla.
                                mover(t,a)
                        Eliminar w de F(v)
        ELSE IF( [nc(v),y(v)+a] =1)
                Sea t el vértice en tal casilla.
                mover(t,a)
```

Mediante la función descrita, se obtiene el conjunto de vértices asociado a cada uno de los dos posibles movimientos de un vértice v. Basta con calcular mover(v,1) para el movimiento a la casilla inferior y mover(v,-1) para la superior.

Así pues, todo vértice v tiene asociados dos movimientos, desplazarse a la casilla inmediatamente superior (o inferior) en su mismo nivel. El algoritmo 6 proporciona el conjunto de vértices que serán desplazados una casilla hacia arriba (o hacia abajo) al mover v.

4.2 EVALUACION DE LOS MOVIMIENTOS

La implementación del primer criterio propuesto la realizaremos de la siguiente manera. Dada una arista (v,w), se tratará de situar los dos vértices de modo que | y(v) - y(w) | sea lo menor posible. Esto equivale a que la arista esté lo más horizontal posible y, de este modo, se minimiza su longitud.

Para considerar todas las aristas incidentes con un vértice v, se define I(v) como el conjunto de vértices adyacentes a v. De este modo se define el evaluador del primer criterio para el vértice v como:

$$e_1(v) = \sum_{w \in I(v)} | y(v) - y(w) |$$

Considerando que en el segundo criterio hay que equilibrar el conjunto de aristas que parten de v y el conjunto de aristas que llegan a v, se definen:

$I_s(v)$ = Conjunto de vértices sucesores de v.
$I_p(v)$ = Conjunto de vértices predecesores de v.

Obviamente $I(v) = I_s(v) \cup I_p(v)$.

Para definir el evaluador del segundo criterio, se consideran los siguientes parámetros:

$$E_s(v) = \left| y(v) - \frac{\sum_{w \in I_s(v)} y(w)}{| I_s(v) |} \right| \qquad E_p(v) = \left| y(v) - \frac{\sum_{w \in I_p(v)} y(w)}{| I_p(v) |} \right|$$

Dado que el equilibrar las aristas sólo tiene sentido cuando hay más de dos sucesores o predecesores, se propone el siguiente evaluador del segundo criterio para el vértice v:

Si $\| I_s(v) \| > 1$ y $\| I_p(v) \| > 1$	entonces	$e_2(v) = E_s(v) + E_p(v)$
Si $\| I_s(v) \| < 2$ y $\| I_p(v) \| > 1$	entonces	$e_2(v) = E_s(v)$
Si $\| I_s(v) \| > 1$ y $\| I_p(v) \| < 2$	entonces	$e_2(v) = E_p(v)$
Si $\| I_s(v) \| < 2$ y $\| I_p(v) \| < 2$	entonces	$e_2(v) = 0$

De este modo, el evaluador de un vértice v se establece como:

$$e(v) = \alpha\, e_1(v) + \beta\, e_2(v)$$

donde $\alpha + \beta$ es constante y las proporciones de ambos introducen el peso que se le quiere dar a cada uno de los criterios. En nuestro caso, consideraremos ambos criterios con igual peso por lo que el evaluador será la suma de los dos evaluadores.

Se define $e(G) = \sum_{v \in V} e(v)$.

Para evaluar un movimiento asociado a un vértice v, se calcula el incremento del parámetro e(G) al realizar el movimiento.

4.3 SELECCION DEL MOVIMIENTO

El objetivo del algoritmo es minimizar el valor de e(G).

Una vez seleccionado un vértice v con el procedimiento especificado en cada fase, se evalúan los dos movimientos asociados a éste.

En la *Fase Improving*, se selecciona el movimiento que más disminuya el valor de e(G). Si ambos hacen aumentar dicho parámetro, no se selecciona ningún movimiento. En la *Fase Mixed*, se selecciona el movimiento con menor incremento de e(G), aún cuando sea positivo.

Considerando el mismo esquema algorítmico que en el apartados anterior, se ha determinado los siguientes valores de los parámetros del algoritmo: $\alpha = 5$, tope1=25, tope2=6*(|M| + |E|) y tope3=25.

5. - COMPARACION CON OTROS ALGORITMOS DE DIBUJO

De todos los trabajos publicados sobre el dibujo automático de sistemas jerárquicos, el artículo más relevante es el de Sugiyama y otros [44].

Comparando el algoritmo presentado en este capítulo con el de dichos autores, podemos observar mejoras notables en el nuestro:

1.- Intersecciones de aristas.

Para minimizar el número de intersecciones Sugiyama y otros [44] utilizan un algoritmo heurístico basado en los baricentros, similar al algoritmo Averaging de Eades y Kelly [9] que a su vez es inferior al algoritmo Splitting de los mismos autores.

Las pruebas computacionales del capítulo tercero ya establecieron la superioridad de nuestro algoritmo Tabu Thresholding.

2.- Aristas de longitud mayor que uno.

Respecto al criterio de que todos los vértices ficticios tengan el mismo valor de ordenada, Sugiyama incorpora este criterio en una segunda fase en la que no permite alterar el orden obtenido en la primera fase de minimización de cortes. Hemos comprobado que dicha implementación no permite, en muchos casos, que se cumpla el criterio mencionado.

En el procedimiento que presentamos, al incorporar el mencionado criterio como una restricción en el algoritmo de minimización, queda garantizado su cumplimiento. Hay que destacar que dicha restricción no hace variar sustancialmente el número de cortes de la solución del algoritmo respecto al algoritmo sin restricciones, tal y como muestra la tabla comparativa mostrada en el apartado 3.6 de resultados.

3.- Paso de Grafo acíclico a Jerarquía.

Al tratar el tema de la asignación de los vértices del grafo a las columnas de una plantilla para definir una jerarquía, Sugiyama cita la solución dada por Warfield [53] en la que la única consideración de dibujo es que todas las aristas sean dibujadas de un nivel a otro superior.

En nuestro algoritmo, además de este criterio, se ha considerado el minimizar la suma de las longitudes de las aristas, resolviendo de manera óptima dicho problema.

4.- Consideraciones computacionales.

Los resultados y ejemplos mostrados por dichos autores son de tamaños pequeños. En nuestro caso, conscientes de la necesidad práctica de resolver problemas reales de gran tamaño, se han diseñado estructuras de datos y códigos de los algoritmo que permiten resolver problemas de gran tamaño en un ordenador personal con bajos tiempos de computación.

6 EJEMPLO

El ejemplo que se muestra, ha sido generado de forma aleatoria. Para ello se ha diseñado un algoritmo que construye un grafo acíclico ordenado topológicamente a partir del número de vértices n y el número de aristas m. En un primer paso garantiza el que todos los vértices tengan, al menos, grado de entrada y salida 1 (excepto el primero y el último). Así, para cada vértice i genera aleatoriamente la arista (i,k) con $i < k \leq n$; después, para cada vértice j al que no le llega ninguna arista genera aleatoriamente la arista (k,j) con $1 \leq k < j$. Una vez conectados todos los vértices, genera aleatoriamente el resto de aristas (i,j) con i<j necesarias para que el número total de aristas sea m.

Con dicho algoritmo, se ha generado un grafo dirigido acíclico de 15 vértices y 25 aristas, del cual hemos realizado manualmente esta primera representación sin considerar ningún criterio de dibujo.

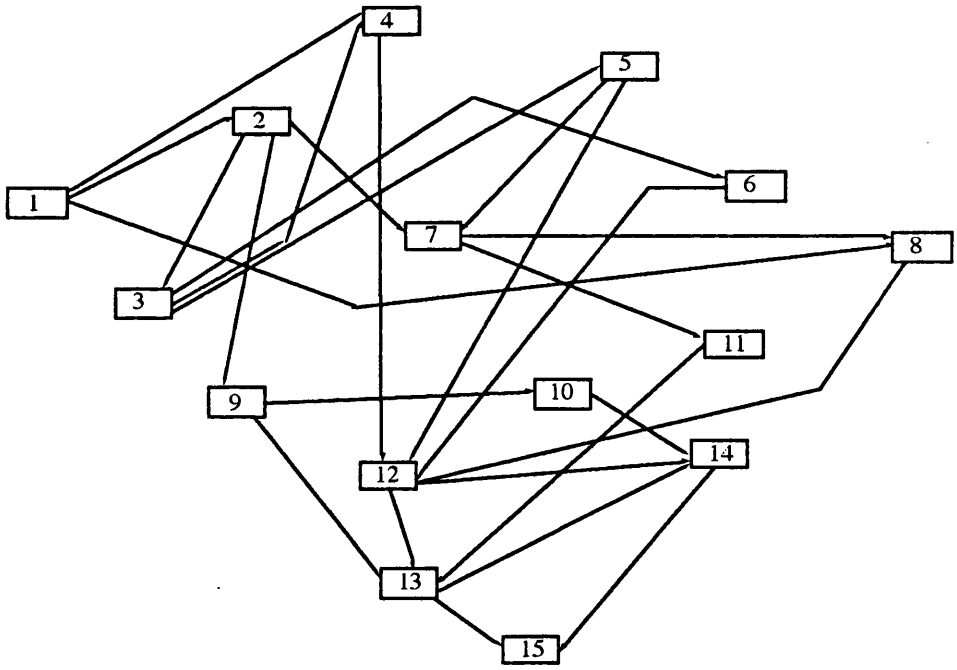


Figura 31

A continuación se muestra cómo el procedimiento algorítmico descrito, va estructurando el grafo hasta producir el dibujo final.

Asignación de vértices a niveles: Jerarquía impropia.

Con el procedimiento mostrado en el primer apartado se calcula el nivel al que hay que asignar cada vértice mediante la resolución de un problema de flujo de coste mínimo, obteniendo la distribución que muestra la figura 32.

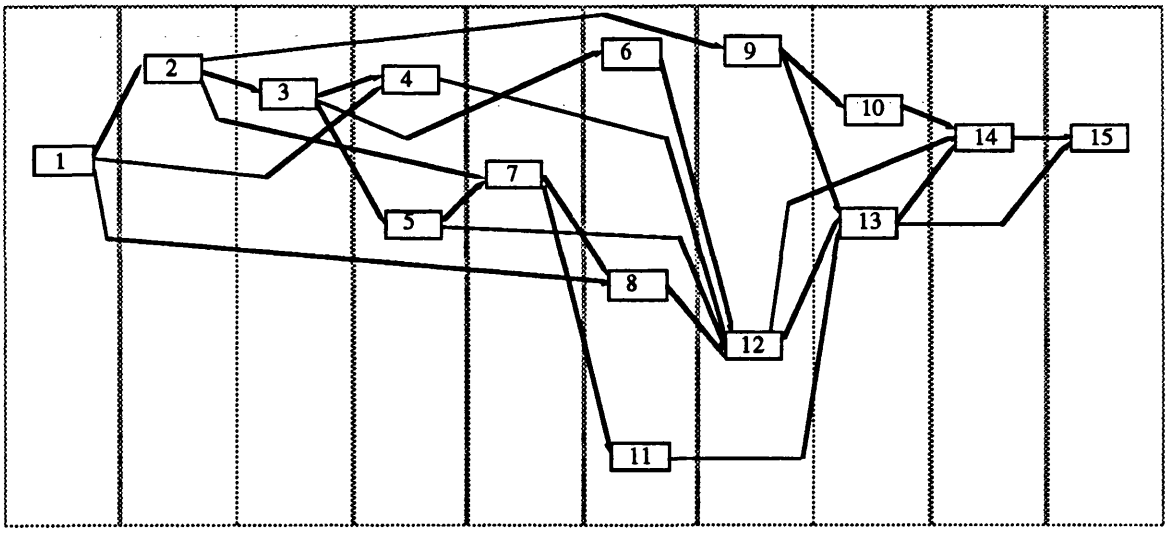


Figura 32

Solución inicial: Hacer la Jerarquía propia.

En primer lugar se identifican las aristas de longitud mayor que uno y se sustituyen por las cadenas de vértices y aristas correspondientes tal y como muestra la siguiente tabla.

Arista	Longitud	Cadena de aristas ficticias
(1,4)	3	(1,16) (16,17) (17,4)
(1,8)	5	(1,18) (18,19) (19,20) (20,21) (21,8)
(2,7)	3	(2,22) (22,23) (23,7)
(2,9)	5	(2,24) (24,25) (25,26) (26,27) (27,9)
(3,6)	3	(3,28) (28,29) (29,6)
(4,12)	3	(4,30) (30,31) (31,12)
(5,12)	3	(5,32) (32,33) (33,12)
(11,13)	2	(11,34) (34,13)
(12,14)	2	(12,35) (35,14)
(13,15)	2	(13,36) (36,15)

Con el algoritmo descrito en el apartado 3.5 se obtiene la distribución inicial cumpliendo el criterio 1 que muestra la figura 33.

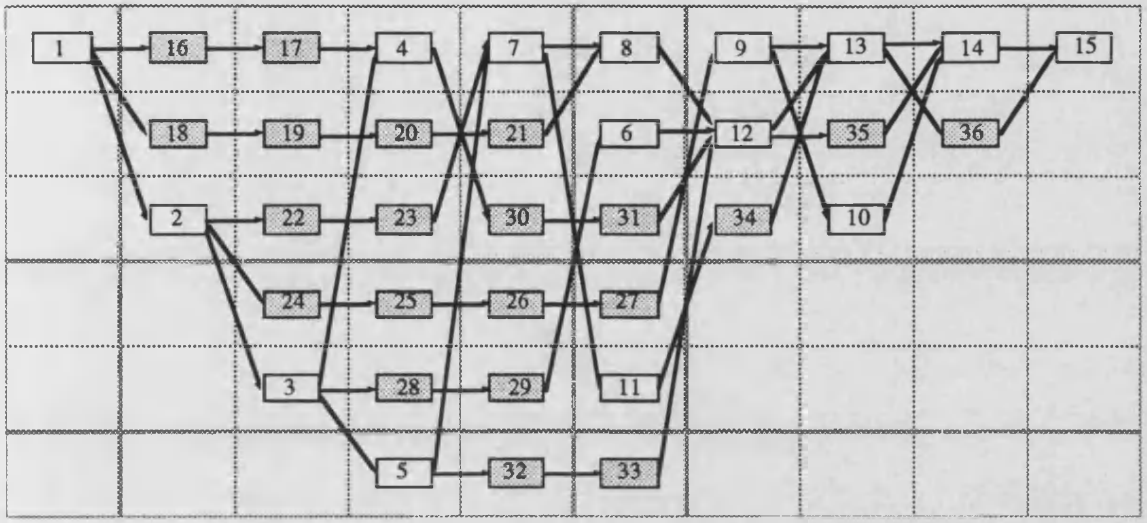


Figura 33

La solución inicial tiene 26 intersecciones de aristas y un valor del parámetro e(G)=70.

Minimización de intersecciones

A partir de la solución inicial y con un parámetro umbral igual a 3, se obtiene la solución que muestra la figura 34 con 3 intersecciones y e(G)=73.

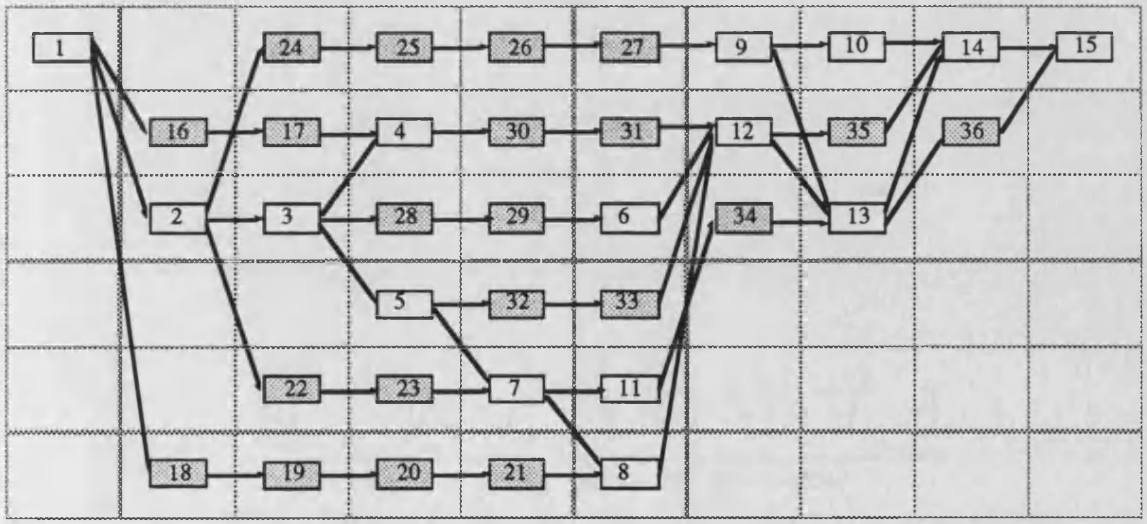


Figura 34

Redistribución Operativa.

Aplicando el segundo algoritmo tabu descrito en este capítulo (apartado 4) a la solución de la figura 34, se obtiene la siguiente solución con el mismo número de intersecciones y con valor de e(G)=52.

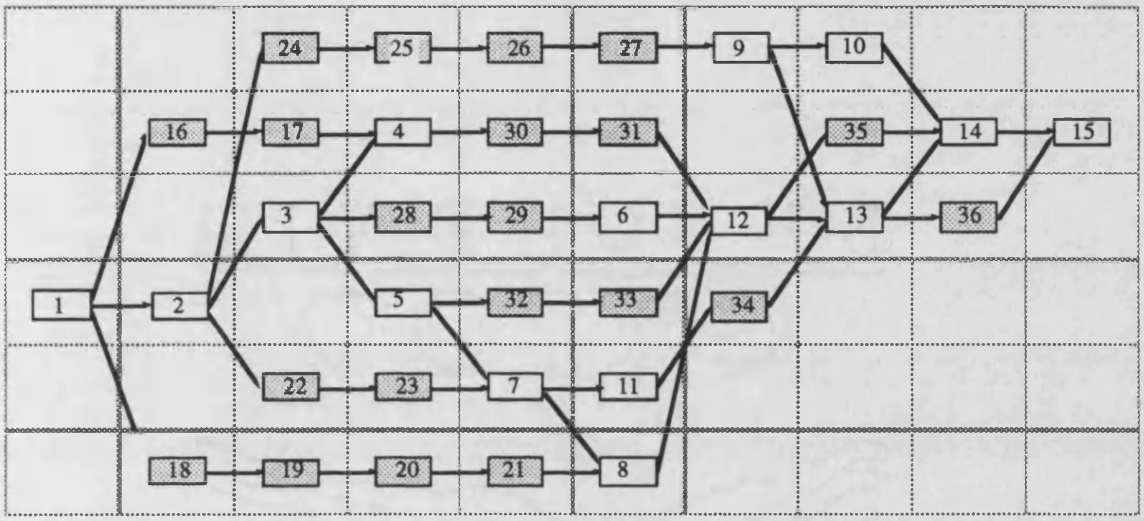


Figura 35

Solución Final

Sustituyendo las cadenas de vértices y aristas ficticias por las originales en el mismo sitio en el que se encuentran éstas, se obtiene la representación final del grafo.

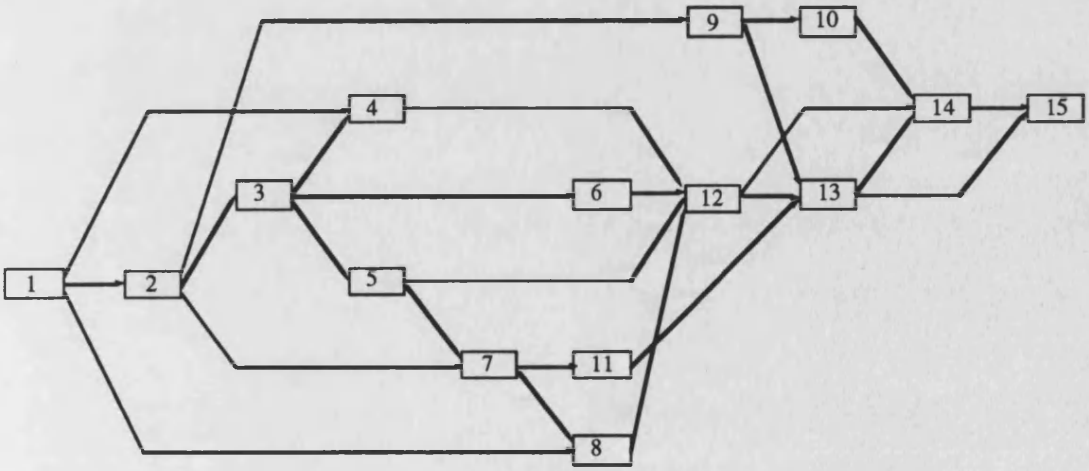


Figura 36

CAPITULO V

Dibujo del Grafo de un Proyecto

En el capítulo anterior se ha visto un algoritmo para dibujar un grafo acíclico. Dicho algoritmo se basaba una serie de criterios para distribuir los vértices del grafo que pueden ser considerados básicos y de aplicación general. El criterio principal era minimizar el número de intersecciones de las aristas y los restantes estaban subordinados a este criterio principal.

Ahora bien, los grafos acíclicos sirven para modelizar sistemas de muy diversa índole. Desde el punto de vista del dibujo, las distintas áreas de aplicación de los grafos acíclicos difieren tanto en la información estructural que pretenden transmitir como en la utilización de los propios dibujos. Por ello, una representación de un grafo acíclico que es buena en un área de aplicación, puede no serlo en otra. Las necesidades específicas de los usuarios de los dibujos en cada área de aplicación han de ser compatibilizadas con los criterios generales de dibujo hasta ahora utilizados.

En los grafos de proyectos de actividades, los vértices representan a las actividades del proyecto estando éstas unidas por unas relaciones tecnológicas. Estas relaciones se traducen en la mayoría de los casos en relaciones de precedencia. Sin embargo, en ocasiones, además de este tipo de relaciones existen otras que no están reflejadas explícitamente en el grafo pero que el planificador ha de tener en cuenta. Para visualizar fácilmente estas segundas relaciones, el planificador agrupa las actividades en bloques o estructuras. Por ello, a nivel operativo es interesante que, al dibujar el grafo, los vértices que representan actividades de un mismo bloque tecnológico aparezcan distinguidos de alguna manera.

Existen dos formas naturales de distinguir, en el plano de un proyecto, un conjunto de tareas:

- Situar una secuencia de actividades "importantes" en el centro del dibujo de manera que sugiera una "escala temporal" del desarrollo del proyecto.

- Situar un grupo de actividades con características comunes en una misma región del plano.

Otro problema que afecta al diseño del plano es que puede interesar el considerar el proyecto a diferentes niveles de detalle o complejidad; esto es: tener una visión de conjunto o ampliar una determinada zona. Así, para ver el proyecto en su totalidad destacando los aspectos importantes se pueden agrupar un conjunto de tareas en una

"super tarea" que modelice un subproyecto. Por ello, el diseño del plano debe permitir expandir un vértice (o colapsar un grupo de vértices) en el proceso de refinamiento y diseño de un proyecto, sin alterar gravemente la apariencia del plano.

En este capítulo se propone una metodología que, fundamentada en todo el trabajo expuesto anteriormente, permita incorporar las necesidades mencionadas introduciendo los conceptos de *Eje Central* y *Componentes* del grafo.

1 EJE CENTRAL DEL GRAFO.

Como se ha mencionado, en algunos casos el planificador quiere que un grupo prefijado de actividades se sitúen a modo de camino o eje central del proyecto. En otros casos, aunque no existe un grupo predefinido, es conveniente que aparezca en el dibujo un eje central de actividades que emule una "escala temporal".

En el trabajo desarrollado vamos a considerar un método que seleccione automáticamente el eje central del grafo. Si existiera un camino de actividades que se desea situar como eje central, se puede incorporar a la metodología que se describe.

En las redes que representan proyectos siempre podemos considerar que existe un vértice inicial (vértice 1) que indica el comienzo de todo el proyecto, y un vértice final (vértice n) que indica el final del proyecto. Como eje central vamos a tomar un **1-n camino más largo** respecto al número de aristas de la red. Tal decisión está basada en las siguientes ideas:

- El camino más largo proporciona el mínimo número de columnas de la plantilla en el que se puede representar la red.

- El considerar el camino más largo como eje de vertebración del plano, permite considerar el plano en toda su extensión desde el inicio del dibujo.

- Por ser un camino todas sus aristas se pueden situar en sentido horizontal de izquierda a derecha.

- Situando como eje central y en posición horizontal dicho camino, queda más explícita la idea de "paso del tiempo de izquierda a derecha", lo cual aportará facilidad de lectura al gráfico resultante.

Sin embargo, en un grafo pueden aparecer varios 1-n caminos más largos, por lo que se ha de establecer un criterio o criterios de selección para escoger uno entre todos ellos (que denominaremos principal y notaremos por P).

Una vez seleccionado el camino más largo P, el siguiente problema es la colocación del resto del grafo respecto a tal camino. Al considerar el grafo resultante de eliminar los vértices de dicho camino junto con las aristas incidentes a ellos, se obtienen una serie de

componentes conexas que aparecen de manera natural como unidades diferentes. Notar que, por el hecho de ser componentes conexas no existen aristas entre ellas y únicamente se consideran las aristas que las conectan con el camino (además de las propias aristas internas de las componentes). Esto proporciona mucha flexibilidad al situar las componentes en la plantilla de trabajo, puesto que sólo se ha de evaluar como quedarán las aristas que las unen con el camino inicial según si la componente se sitúa en una posición o en otra.

Respecto a la elección del camino P podemos considerar dos posibilidades:

- Tomar el 1-n camino más largo que más desconecte el grafo. Y por lo tanto, que mayor número de componentes conexas produce al eliminarlo del grafo.

- Tomar el 1-n camino que menos componentes conexas produzca al eliminarlo del grafo.

La primera elección permite mayores posibilidades al diseñar el grafo, dada la existencia de más componentes conexas. La segunda es conveniente en la medida en que sea bueno el método que se va a aplicar en cada componente, ya que previsiblemente aparecerán pocas.

Considerando que, en general, en un grafo los vértices con mayor grado son los que al eliminarlos más lo desconectarán (y los de menor grado los que menos), éste será el criterio de selección del camino.

El procedimiento para seleccionar cualquiera de ambos caminos tiene dos fases. En la primera fase se construye el *subgrafo de caminos más largos*. En la segunda se escoge de dicho grafo el camino.

Definición

Dado el grafo G=(V,E) se define el subgrafo de caminos más largos G´=(V´,E´) del siguiente modo:

V´= Conjunto de vértices de G tal que cada uno se encuentra al menos en un camino más largo de G.

E´= Conjunto de aristas de G tal que cada una se encuentra, al menos, en un camino más largo de G

TEOREMA 1

Todo 1-n camino en el subgrafo G´es un 1-n camino más largo en G.

Prueba

Dado el grafo G=(V,E), sea u(v) el número de aristas en el camino más largo desde el vértice 1 hasta v en G.

Puesto que toda arista de G´va de un vértice a otro con u(j) una unidad mayor, u(n) es fijo y u(1)=0, todo 1-n camino en G´tiene necesariamente la longitud de un camino más largo en G.■

Para obtener el subgrafo G´se propone el siguiente algoritmo.

Algoritmo 1. Subgrafo de caminos más largos

Inicialización

cont=1
u(v)=0 $\forall v \in V$
v´= B = {n}
A´= ∅

Paso hacia adelante

```
WHILE (cont < n)
        v=cont
        ∀ w / (v,w) ∈ A  hacer:
                IF( u(w) < u(v) +1)
                        u(w) = u(v) +1
        cont = cont+1
```

Paso hacia atrás

```
WHILE( B≠∅ )
        Escoger v de B y hacer:
                B = B - {v}
                ∀ w ∈ V / (w,v) ∈ A y u(w) = u(v)-1 hacer:
                        V´= V´ ∪ {w}
                        B = B ∪ {w}
                        A´= A´ ∪ {(w,v)}
```

Una vez calculado G´se considera un problema de **flujo de coste mínimo y valor 1** en el grafo G´ con los siguientes parámetros:

Vértices.

Vértice 1 con oferta 1

Vértice n con demanda 1.

Resto de vértices de transbordo.

Aristas.

Consideramos las cotas de todas las aristas (l,u)=(0,1).

El coste de cada arista será el grado en G del vértice al que llega cambiado de signo[1].

Es claro que al mandar una unidad de flujo desde 1 hasta n, buscará el 1-n camino de coste mínimo, y por ser los costes de las aristas el grado del vértice al que llegan, esto nos dará la solución.

[1] Si se cambia el signo, como el flujo es de coste mínimo, encontrará el 1-n camino que más desconecta el grafo. Para obtener el que menos desconecta, basta con no cambiar dicho signo.

2 COMPONENTES.

El esquema algorítmico que se propone, permite agrupar un conjunto de vértices (componente) en una misma región del plano. Como ya se ha visto, al calcular el camino principal P y eliminarlo del grafo G, aparecen una serie de componentes conexas que serán consideradas hasta cierto punto de forma independiente.

Si el planificador quiere que un conjunto W de vértices aparezca agrupado, se procederá de la siguiente manera:

Dado que, al calcular el camino P es preferible que en tal camino no se tome ningún vértice de W, se seguirá el siguiente esquema:

1 - Hallar G´.
2 - Poner coste $+\infty$ a las aristas incidentes con vértices de W.
3 - Al resto de aristas poner coste del grado en G del vértice al que llegan.
4 - Calcular un flujo de coste mínimo en el grafo resultante.

Si con dicho procedimiento se obtiene un camino P en el que no hay ningún vértice de W, las componentes conexas se calculan en el grafo G-P-W en lugar de hacerlo en el grafo G-P. De esta manera, se obtiene por un lado el camino P y por otro una serie de componentes conexas a las que podemos añadir W.

Si en dicho camino aparece algún vértice de W, se considerarán P y W conjuntamente al comenzar el dibujo.

En caso de no que no exista ningún conjunto especial de vértices, se calcularán las componentes conexas del grafo G-P.

Para obtener tales componentes conexas, se utiliza un algoritmo de tipo recursivo que realiza una exploración D.F.S. sobre los vértices del grafo. Se considera el grafo acíclico G=(V,E) donde V={1,2, ..., n} y se define para cada vértice v el parámetro nc(v) que es igual al número de la componente conexa de G-P a la que pertenece v.

El algoritmo comienza asignando a todos los vértices $v \in V$ un valor de nc(v)=0, salvo para los vértices de P a los que les asigna un valor de nc(v)=-1. El algoritmo examina todos los vértices de G cuando encuentra un vértice v que no pertenece a P y no

ha sido explorado (nc(v)=0), se pasa el vértice v a la función "explorar", la cual recursivamente visita todos los vértices conectados al dado, asignándoles el mismo número de componente que el que asigna a v.

Puesto que el grafo es dirigido, a la hora de explorar los vértices adyacentes a un vértice dado, se distingue entre sucesores y predecesores.

Algoritmo 2. Componentes conexas

Inicialización.

```
V={1,2, ..,n}
i=1.
c=0
nc(v)=-1  ∀ v ∈ P
nc(v)= 0  ∀ v ∈ G-P
```

Iteraciones.

```
While( i ≤ n)
        v=i
        IF( nc(v) = 0)
                c = c+1
                Explorar(v)
        i=i+1.
```

Función auxiliar.

```
Explorar(v)
        nc(v)=c.
        ∀ w ∈ Sucesores(v) / nc(w) = 0
                        explorar(w)
        ∀ w ∈ Predecesores(v) / nc(w) = 0
                        explorar(w)
```

3 DESCRIPCION DEL ALGORITMO

Al igual que se ha hecho en el capítulo anterior se considera el grafo sobre una plantilla de modo que queden definidos unos ejes coordenados que permitan referenciar fácilmente los vértices.

El número de columna al que se asigna cada vértice se determina como se mostró en el primer apartado del capítulo anterior, de forma que la suma de las longitudes de las aristas sea mínima.

El esquema algorítmico es el siguiente:

1 - Determinar el camino principal P.
2.- Calcular las componentes conexas que aparecen al eliminar el camino del grafo.
3.- Dibujar el camino P y las aristas en dicho camino.
4.- Mientras queden componentes por asignar:
    4.1 - Ubicar cada componente en una región del plano.
    4.2 - Situar los vértices y trazar las aristas dentro de cada componente.
    4.3.- Trazar las aristas que unen cada componente con el camino P.

Los puntos 1 y 2 han sido estudiados en los apartados anteriores. Veamos los puntos siguientes:

3. Dibujar el camino P y las aristas en dicho camino.

Se sitúa el camino P en la plantilla. Se trazan las aristas de longitud uno que unen los vértices contiguos y el resto de aristas se ordena por orden creciente de longitud. Con este orden se van asignando las aristas arriba o abajo de P donde menos intersecciones generen, trazándolas en tres tramos como se describió en el apartado segundo del capítulo anterior (ver figura 1)

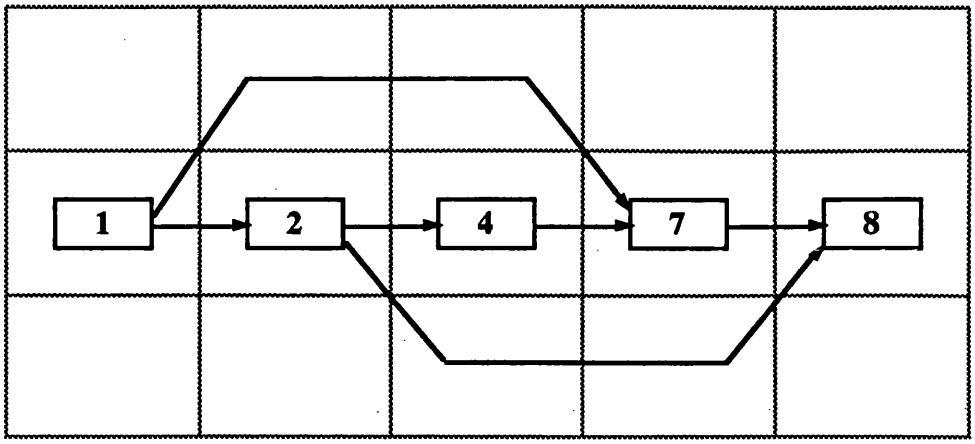


Figura 1. Trazado de aristas de P

4. Situación de las componentes.

Las componentes se ordenan según el número de aristas con el camino P por orden creciente.

Cada componente se sitúa arriba o abajo del camino P en la fila más cercana a P posible. Dicha elección se realiza considerando el sitio donde las aristas entre la componente y P producirán menos intersecciones con las aristas ya dibujadas.

La distribución de los vértices y el trazado de las aristas de cada componente conexa (4.2) se realiza con el algoritmo de dibujo presentado en el capítulo anterior.

Para resolver el problema del trazado de las aristas que unen los vértices de las componentes con los vértices del camino P, se presenta un procedimiento basado en las siguientes definiciones:

Definición

Sea un grafo acíclico G=(V,E) en el que los vértices han sido asignados a columnas definiendo una jerarquía. Sea P un 1-n camino más largo en G.

Se define la jerarquía G´ como el grafo que se obtiene al sustituir en G las aristas de longitud mayor que uno, cuyos extremos no pertenecen ambos a P, por cadenas de vértices y aristas de longitud uno.

Se consideran las componentes conexas de G´-P, que coincidirán con las componentes conexas de G-P a las que se le han añadido algunos vértices ficticios. A cada una de dichas componentes conexas, se le aplica el algoritmo de dibujo presentado en el capítulo anterior. De este modo, una vez situada la distribución final de la componente conexa junto al camino P en la plantilla, se tiene que, las aristas de longitud mayor que uno que unen P con la componente, lo hacen con vértices ficticios, por lo que, su trazado ya ha sido tenido en cuenta al distribuir los vértices de la componente. Esto unido al hecho de que el resto de aristas que unen P con la componente conexa son de longitud uno y por lo tanto su trazado trivial, hacen que mediante este procedimiento esté resuelto el paso 4.3 del algoritmo.

4 EJEMPLOS

Ejemplo 1

Consideremos el mismo ejemplo que se generó aleatoriamente en el capítulo anterior.

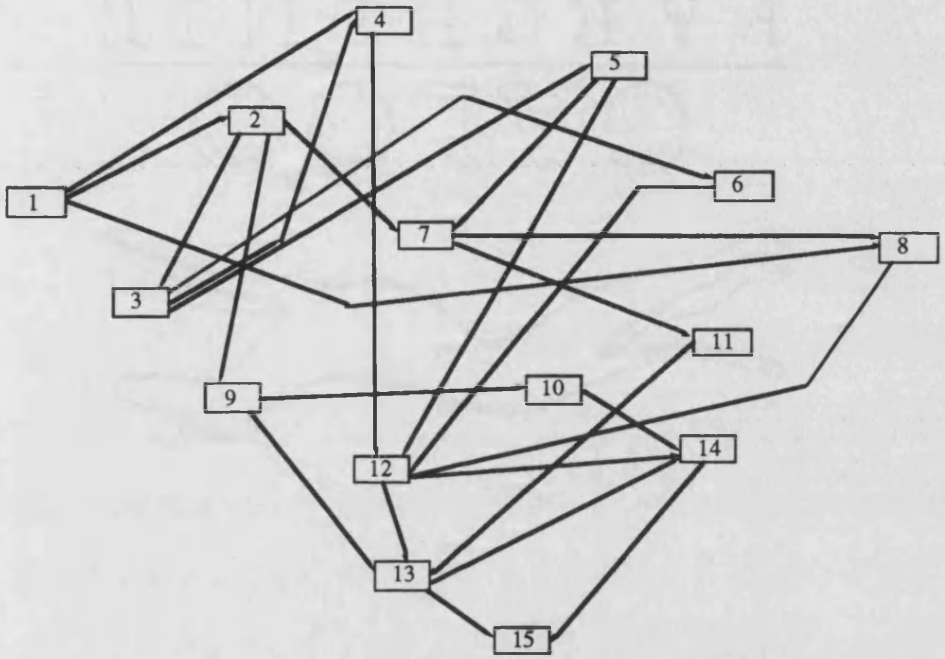


Figura 2. Grafo original

Se calcula el número de columna al que se asigna cada vértice de igual modo al que se detalló. En este grafo sólo existe un 1-15 camino más largo, que es el:

1 - 2 - 3 - 5 - 7 - 8 - 12 - 13 - 14 - 15

La siguiente figura muestra la ubicación del camino P y el trazado de las aristas en dicho camino.

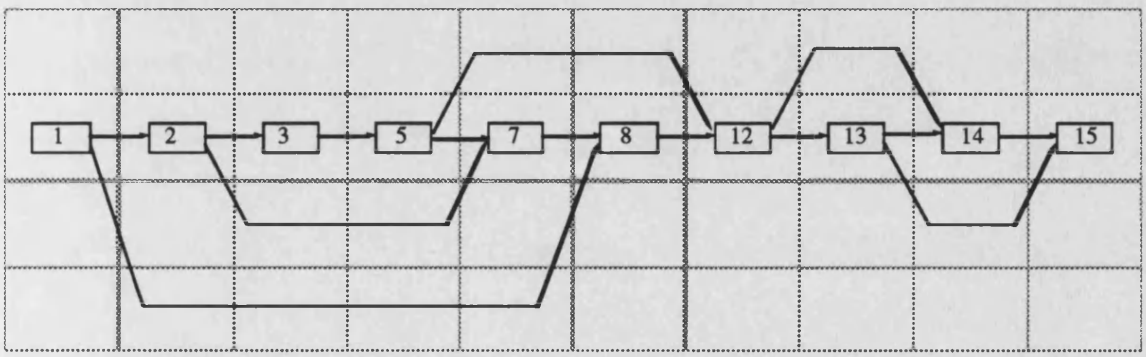


Figura 3. Camino P

Al eliminar el camino P de G aparecen las siguientes componentes conexas, que ordenadas según el número de aristas con P quedan:

{ 6 } , { 11 } , { 9 , 10 } , { 4 }

En dichas componentes las aristas de longitud mayor que uno (tanto internas como las que le unen con P) se sustituyen por cadenas de vértices y aristas ficticias.

A cada componente se le aplica el algoritmo del capítulo anterior. En este caso, dado el tamaño, el resultado es trivial.

Se comienza asignado la componente formada por el vértice 6 y los dos ficticios que modelizan a la arista (3,6) encima del camino P en la menor fila posible: segunda fila. Notar que asignar la componente debajo de P (tercera fila por debajo) generaría dos intersecciones.

Con el procedimiento descrito, se obtiene el siguiente grafo final:

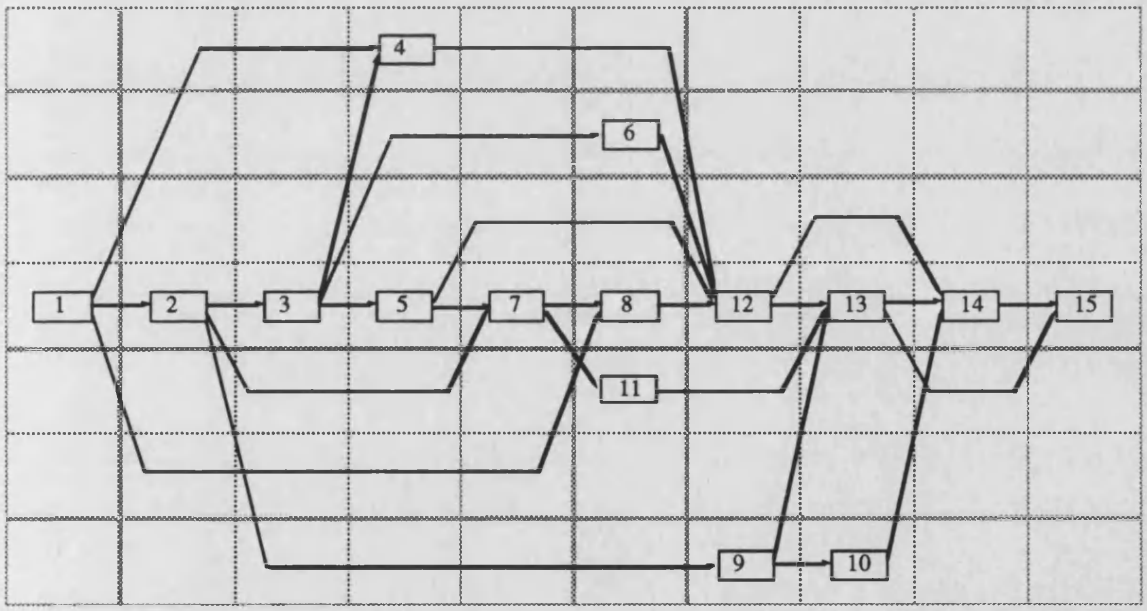


Figura 4: Grafo final

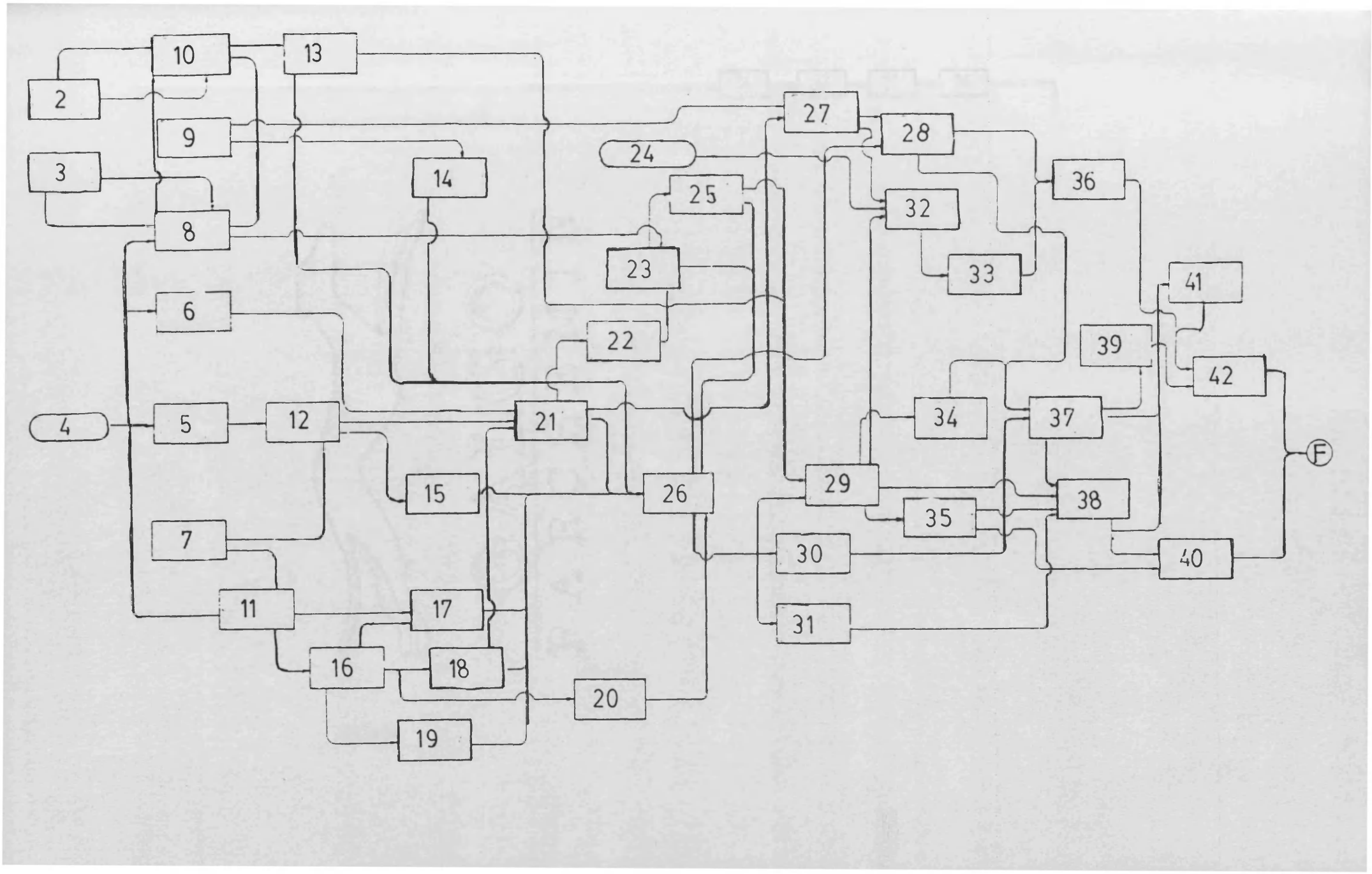
Ejemplo 2

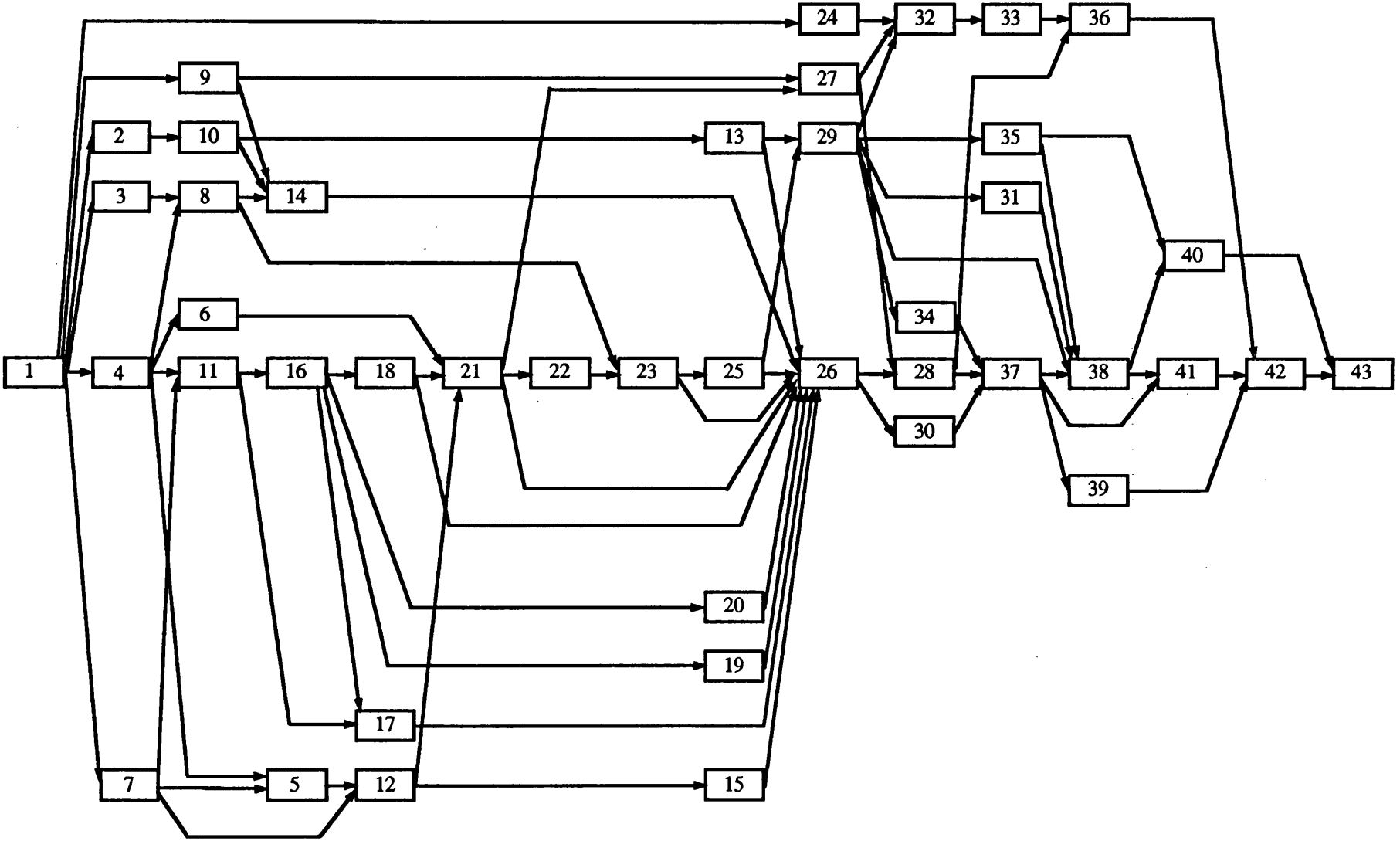
En este caso se considera un ejemplo real. La figura de la página siguiente (plano 1) es un plano de un proyecto tomado del departamento de planificación de una empresa de la Comunidad Valenciana.

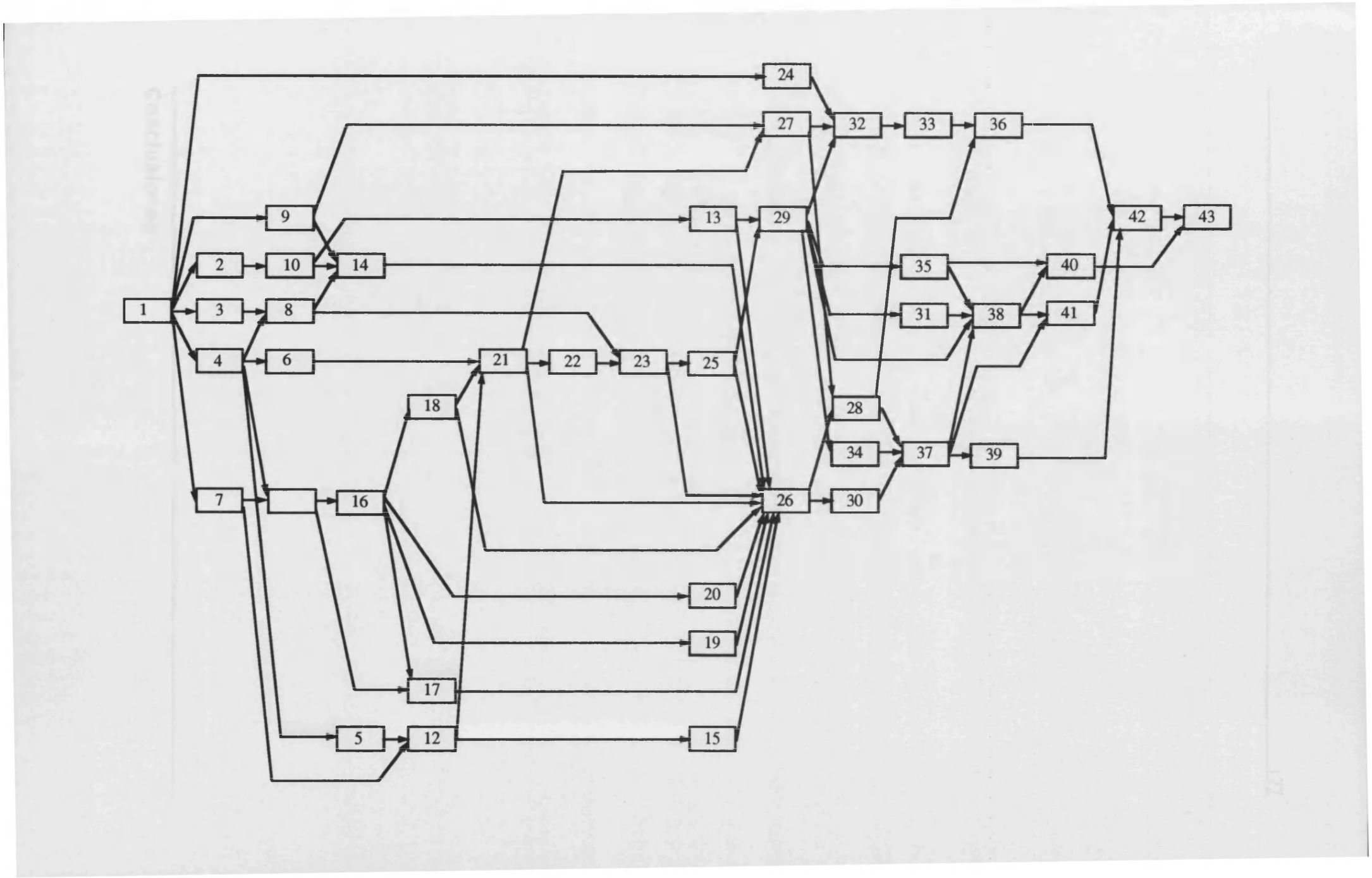
A continuación se muestra el grafo obtenido de modo automático por el algoritmo presentado en este capítulo (plano 2). En dicho grafo se ha tomado como camino P el que más desconecta al original. Tras éste, se adjunta el obtenido con el algoritmo del capítulo anterior (plano 3).

A simple vista se aprecia que los planos 2 y 3 son mucho mejores en cuanto a operatividad y claridad que el plano 1, además de por el hecho de haber sido realizados automáticamente.

El plano 2 tiene 21 intersecciones de aristas, mientras que en el plano 3 aparecen 20. Notar que en el plano 1, al compartir las aristas tramos de líneas, el número de intersecciones que se puede contar no es el real, ya que habría que representar a cada arista por una línea independiente para que se pudiera medir dicho número. Aún así, el número que aparece es de 35.


10
13
2
9
27
28
24
3
14
25
36
32
8
23
33
41
6
22
39
42
4
5
12
21
34
37
F
15
26
29
38
7
35
30
40
11
17
31
16
18
20
19




1
2
3
4
7
9
10
8
6
14
16
18
21
22
23
25
13
29
24
27
32
33
36
35
31
38
40
41
42
43
28
34
37
39
26
30
20
19
17
5
12
15

La temática abordada en esta memoria es la referente al dibujo automático del grafo de un proyecto

El trabajo se ha dividido en dos partes.

En la **primera parte** se ha estudiado la minimización del número de las intersecciones de las aristas del grafo, por considerar este criterio el más importante y complejo al representarlo.

Para resolver este primer problema se han diseñado dos algoritmo: uno exacto y otro heurístico.

En el **capítulo segundo** comenzamos estudiando el caso particular de grafos bipartidos, que aún siendo más simple, es NP-completo (Eades [11]). Así, se presenta un algoritmo exacto basado en las técnicas de "Branch & Bound", que según nuestras referencias bibliográficas, es el primer algoritmo exacto diseñado para resolver el problema planteado.

En el apartado de resultados se muestra el comportamiento del algoritmo sobre una colección de cuatrocientos grafos bipartidos generados aleatoriamente y clasificados según su densidad. La eficacia del algoritmo es manifiesta, a la vista de los resultados,tanto por los bajos tiempos computacionales como por el pequeño número de nodos explorados en el árbol de soluciones con relación al tamaño de los problemas.

En la segunda sección del capítulo, se generaliza el algoritmo exacto a grafos acíclicos cualesquiera. Sin embargo, tal generalización incrementa en gran medida el número de soluciones posibles, haciendo impracticable resolver problemas reales de manera exacta.

En el **capítulo tercero** se desarrolla un algoritmo heurístico, basado en las técnicas de optimización "Tabu Search", para resolver el problema citado.

En el primer apartado vuelven a considerarse grafos bipartidos. Los resultados del algoritmo sobre este tipo de grafos han sido comparados con la solución óptima, cuando se disponía de ella. Asimismo, se han contrastado los resultados obtenidos con los algoritmos Greedy Switching y Splitting debidos a Eades y Kelly [9], por considerarlos los mejores publicados.

Al trabajar sobre una colección de doscientos cincuenta grafos bipartidos generados aleatoriamente se ha visto que: en todos los casos en los que se disponía del óptimo, la solución del algoritmo heurístico era la óptima. Asimismo, en todos los ejemplos probados la solución del algoritmo tabu mejoró a la de los otros dos algoritmos considerados.

Considerando la solución obtenida con el algoritmo heurístico, se estudia el funcionamiento del algoritmo exacto del capítulo anterior, utilizando como cota superior inicial el número de intersecciones de la solución tabu. De este modo, se muestra como los tiempos de computación del exacto se reducen en torno a un 25%.

En el segundo apartado se generaliza el algoritmo heurístico a grafos acíclicos cualesquiera. Para medir la eficacia del algoritmo se ha trabajado con una colección de grafos de hasta seiscientos vértices, con tiempos de ejecución siempre inferiores a los dieciséis minutos. La relación del algoritmo con los trabajos más relevantes publicados sobre este tema es tratada, mostrando la superioridad del algoritmo presentado.

En la **segunda parte** de la memoria se ha estudiado el problema del dibujo de grafos acíclicos.

En el **capítulo cuarto** se ha presentado un proceso algorítmico que obtiene una representación del grafo en tres fases considerando los siguientes criterios de dibujo:

1 - Trazar las aristas de izquierda a derecha, nunca de derecha a izquierda.(Martí[34])
2 - Minimizar el número de cruces de aristas. (Eades y Kelly [9])
3 - Utilizar el mínimo número de columnas de la plantilla.
4 - Situar los vértices adyacentes en posiciones cercanas. (Sugiyama y otros [44])
5 - Reducir la suma de las longitudes de las aristas (Eades y Wormald [13],Tagawa [45])
6 - Dibujar claramente las estructuras de ramificación y unión.(Sugiyama y otros [44])
7 - Evitar una excesiva densidad de información. (Carpano [3])
8 - Trazar las aristas mediante poligonales de tres lados o menos.

En la fase 0 se asignan los vértices del grafo a las columnas de la plantilla considerando el optimizar los criterios 3 y 5[1].

[1] Se considera que la longitud de una arista es la diferencia entre número de columna de su vértice final y el inicial.

En la fase 1 se reordenan los vértices del grafo de modo que se optimice el criterio 2 y se respete la restricción enunciada en el criterio 8 reforzando subsidiariamente el criterio 5. Para ello, se introduce dicha restricción en el algoritmo tabu de minimización de cortes desarrollado en el capítulo anterior, lo cual añade bastante complejidad al conjunto de movimientos.

Para medir la eficiencia del algoritmo presentado en esta fase, así como el impacto de la restricción introducida, se compara éste con el del capítulo anterior, apreciándose un pequeño aumento en el número de intersecciones que, a nuestro juicio, queda más que compensado por el incremento de la calidad global del dibujo.

En la fase 2 se desarrolla otro algoritmo heurístico, basado también en las técnicas "Tabu search", que reubica los vértices de cada columna sin alterar las ordenaciones obtenidas en la fase anterior, de modo que se optimicen los criterios 4, 6 y 7 y no se pierda la bondad de la solución respecto a los criterios considerados anteriormente.

Para ilustrar el comportamiento del proceso, se muestra cómo va estructurando en cada fase a un grafo acíclico dirigido, generado aleatoriamente, hasta obtener su representación final. El algoritmo se compara con el presentado por Sugiyama y otros [44] por considerarlo el más relevante aparecido en la literatura científica, mejorándolo en todos los criterios considerados.

En el **capítulo quinto** se propone una metodología que fundamentada en todo el trabajo expuesto anteriormente, permita considerar necesidades específicas de los gestores que representan a proyectos; los conceptos de *Eje Central* y *Componentes* del grafo establecen un marco de trabajo en el que incorporar algunas de esas necesidades. El funcionamiento del algoritmo es mostrado sobre el mismo ejemplo aleatorio considerado en el capítulo anterior. Asimismo, se considera un ejemplo real del plano de un proyecto de una empresa, contrastando el documento original facilitado por ésta con los dos grafos obtenidos con los algoritmos de los capítulos cuarto y quinto.

Como conclusión final consideramos que el trabajo realizado en esta memoria cumple los objetivos enunciados en la introducción, en la medida en que se presentan unas técnicas automáticas de dibujo que, avaladas por los desarrollos matemáticos y por las pruebas computacionales realizadas, mejoran globalmente a los algoritmos publicados más relevantes.

Una futura línea de trabajo consistirá en profundizar la utilización de los conceptos de Eje central y Componentes.

Una operación muy útil al trabajar con planos de proyectos consiste en sustituir un conjunto de actividades (o componente) por un supervértice que las represente. Para trabajar matemáticamente con esta operación habría que definir formalmente la operación de colapsar, caracterizando los subgrafos colapsables. Asimismo, desde el punto de vista práctico es conveniente que al colapsar un conjunto de vértices no cambie drásticamente la apariencia del dibujo. Habría que desarrollar la idea de estabilidad de un dibujo.

La extensión del trabajo realizado en esta memoria a grafos dirigidos con ciclos y a grafos mixtos, constituirá otra línea de investigación que ampliaría su ámbito de aplicación.

[1] **Bazaraa M.S.** y **Jarvis J.J.**, Programación lineal y flujos en redes, Ed. Limusa, México, 1984.

[2] **Bondy J.A.** and **Murty U.S.R.** , Graph theory with Applications, The Macmillan press LTD, Hong Kong, 1978

[3] **Carpano M.J.**, "Automatic Display of Hierarchized graphs for computer aided decision analysis",IEE transactions on systems ,man, and cybernetics, vol SMC-10, no 11, pp 705-715, 1980

[4] **Consuegra J.**, "Software de control de proyectos", Binary n. 17, 1990.

[5] **Christofides N.**, Graph Theory, An algorithmic approach, Academic Press, London 1975.

[6] **Dilworth R.P.**, "A descomposition theorem for partially ordered sets", Ann. of Math, Vol 51, 161-166, 1950.

[7] **di Battista G.** and **Tamassia R.**, "Algorithms for plane representations of acyclic digraphs", Theoretical Computer Science 61, pp 175-198, 1988.

[8] **Dressel G.**, "Medios de organización de la empresa constructora", Colección Organización de la construcción Tomo 2. Editores técnicos asociados, Barcelona 1976.

[9] **Eades P.** and **Kelly D.**, "Heuristics for drawing 2-layered networks", Ars combinatoria 21, pp 89-98, 1986.

[10] **Eades P.** and **Lin X.**, "How to draw a directed graph", Technical Report, Departament of computer Science, University of Queensland, Australiza 1989.

[11] **Eades P.**, **McKay B.** and **Wormald N.C.**, "An NP-complete crossing number problem for bipartite graphs", 1986

[12] **Eades P.**, **McKay B.D.** and **Wormald N.C.**, "On an edge crossing problem", Proceedings of the Ninth Australian Computer Science Conference, Australian Natioonal University, Canberra, pp327-334.

[13] **Eades P.**and **Wormald N.C.**, "The mediam heuristic for drawing 2 - layered network", Technical report 69, Dept. of computer science, University of Queesland, 1986.

[14] **Fary I.**, "On straight line representation of planar graphs", Acta Sci. Math. Szeged 11, 229-233, 1948.

[15] **Fischer M.J.** and **Paterson M.S.** , "Optimal tree layout", Proceedings, ACM Stoc Conference, Los Angeles, 177-189, 1980.

[16] **Floyd R.W.**and **Ullman J.D.**, "The compilation of regular expressions into integrated circuits", Tech Report, Standford University, 1980.

[17] **Glover F.**, "Candidate List strategies and Tabu Search", Research Report, School of Business, University of Colorado, Boulder, Junio 1989.

[18] **Glover F.**, "Tabu Search: A Tutorial", Research Report, School of Business, University of Colorado, Boulder, 1990.

[19] **Glover F.**, "Tabu Search: Part I", ORSA Journal on Computing 1, pp 190-206, 1989

[20] **Glover F.**, "Tabu Search: Part II", ORSA Journal on Computing 1, pp 4-32, 1990.

[21] **Glover F.**, "Simple Tabu Thersholding in optimization", Research Report, School of Business, University of Colorado, Boulder, Septiembre 1991.

[22] **Harary F.**, Graph Theory, Addison Wesley Publishing Company, 1969.

[23] **Hopcroft J.** and **Tarjan R.**, "Efficient planarity testing", Journal of the Association of Computing. Machinery, vol 21-4, pp 549-568, 1974.

[24] **Hope A.K.** , "A planar graph drawing program", Software practice and experience, vol 1, pp 83-91, 1971.

[25] **Johnson D.S.**, "The NP-completeness column: an ongoing guide", Journal of Algorithms, Volume 3, p 97, 1982

[26] **Kennington J.L.** and **Helgason R.V.**, Algorithms for network programming, John Wiley, 1980

[27] **Klovstad P.**, "The technological network; Macro and micro structures", Proceedings of the second international congress Amsterdam, The Netherlands, 1969.

[28] **Knuth D.E.**, The art of computer programming: Fundamental Algorithms, Addison-Wesley, 1973.

[29] **Knuth D.E.**, The art of computer programming: Sorting and Searching, Addison-Wesley, 1973.

[30] **Laguna M.** and **Glover F.**, "On target Analysis and Diversification in Tabu Search", Research Report, School of Business, University of Colorado, Boulder, Junio 1989.

[31] **Lempel A.** and **Cederbaum I.**, "Minimun feedback arc and sets of directed graph", IEEE Trans. Circuit Theory, vol ct-13,n. 4, pp 399-403, 1966.

[32] **Leung J.**, "A new graph theoretic heuristic for facility layout", Mangement Science, Vol 38, n 4, 594-605, 1992.

[33] **Malone D.W.**, "An introduction to the application of interpretive structural modeling", Proc. IEEE Vol. 63, 3, PP 397-404, 1975.

[34] **Martí R.**, Algoritmos para gráficos en la planificación de proyectos, Memoria de licenciatura, Dpto de Estadística e I.O., Universidad de Valencia, 1991.

[35] **Michalewizc Z.**, Genetic Allgorithms + Data structures = Evolution Programs,Springer Verlag, 1992.

[36] **Moder J.J., Phillips C.R.** and **Davis E.W.**, Project Management with CPM, PERT and Precedence Diagramming, Van Nostrand Reinhold Company, New York, 1964.

[37] **Otten R.H.J.M.** and **van Wijk J.G.** , "Graph representation in interactive layout design", Proc. IEEE International Symposium on Circuits and systems, New york, pp 914-918, 1978.

[38] **Reingold E.M., Nievergelt J.** and **Deo N.**, Combinatorial Algorithms: Theory and practice, Ed. Prentice Hall 1977.

[39] **Rosemberg A.L.**, "ON embedding graphs in grids", IBM Report RC-7559, 1979.

[40] **Smith J.** and **Linders J.**, "Automatic generation of logic diagrams", IEEE Proc. 13th Design Automation Conf., n 76, ch. 1098-3c, pp 377-391, 1976.

[41] **Stein S.K.**, "Convex maps", Proc. Amer. Math Soc. 2, 464-466, 1951.

[42] **Storer J. A.** , "The node cost measure for embedding graphs in planar grid", Proc. 12th ACM Symposium on the theory of Computing,pp 201-210, 1980.

[43] **Storer J.A.** , "On minimal node-cost planar embeddings", Networks 14, pp 181-212, 1984.

[44] **Sugiyama K., Tagawa S.** and **Toda M.** "Methods for visual understanding of Hierarchical system structures",IEE transactions on systems ,man, and cybernetics, vol SMC-11,no 2, pp 109-125, 1981

[45] **Sugiyama K., Tagawa S.** and **Toda M.**, "Effective representations of hierarchical structures", International Institute for advanced study of social information science, Fujitsu Limited, Research Report n. 8, 1-29, 1979.

[46] **Tamassia R.** and **Tollis I. G.**, "Plane representation on graphs and visibility between parallel segments", Tech. Report ACT-57, Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, Urbana, IL, 1985.

[47] **Tamassia R.** and **Tollis I.G.** , "A unified approach to visibility representations of planar graphs", Discrete & Computational Geometry 1, pp321-341, 1986.

[48] **Tamassia R., Di Battista G.** and **Batini C.**, "Automatic graph drawing and readability of diagrams", IEEE transactions systems, man and cybernetics, vol 18-1,pp 61-79, 1988.

[49] **Tamassisa R.**, "On embedding a graph in the grid with the minimun number of bends", SIAM J. Comput 16,pp 421-444,1987.

[50] **Tutte W.T.** , "Convex representations of graphs", Proc. London Math Soc. 10, pp 304-320, 1960.

[51] **Tutte W.T.** , "How to draw a graph", Proc. London Math Soc. 3, pp 743-768, 1963.

[52] **Wagner K.**, "Bemerkungen zum Vierfarbenproblem", Jber. Deutsch, Math-verein 46, 26-32, 1936.

[53] **Warfield J.N.** , "Crossing theory and hierarchy mapping," IEE transactions on systems ,man, and cybernetics, vol SMC-7,no 7, pp 505-523, 1977.

[54] **Warfield J.N.**, "Binary matrices in system modeling", IEEE Trans. Syst. , Man, Cybern., Vol SMC-3, pp 441-449, 1973.

[55] **Warfield J.N.**, Societal Systems: Planning, Policy and Complexity. New york: John Wiley and sons, 1976.

[56] **Warfield J.N**, "Extending interpretive structural modeling", Proc. 7th Annual Pittsburgh Conf. Modeling and Simulation, pp 1163-1167, April 1976.

[57] **Wiest J.D.** and **Levy F.K.**, A management guide to PERT/CPM with GERT/PDM/DCPM and other networks, Prentice-Hall, New Jersey, 1977.

[58] **Woods D.**, Drawing Planar Graphs, P.h. D. Thesis, Computer Science Dept., Standford University, 1982.

UNIVERSIDAD DE VALENCIA FACULTAD DE CIENCIAS MATEMÁTICAS

Reunido el Tribunal que suscribe, en el día de la fecha, acordó otorgar, por unanimidad, a esta Tesis doctoral de D. RAFAEL MARTI CONQUERO

la calificación de APTO CUM LAUDE

Valencia, a 4 de JUNIO de 1993

El Presidente

El Secretario,

UNIVERSITAT DE VALÈNCIA — BURJASSOT — FACULTAT DE CIÈNCIES MATEMÀTIQUES